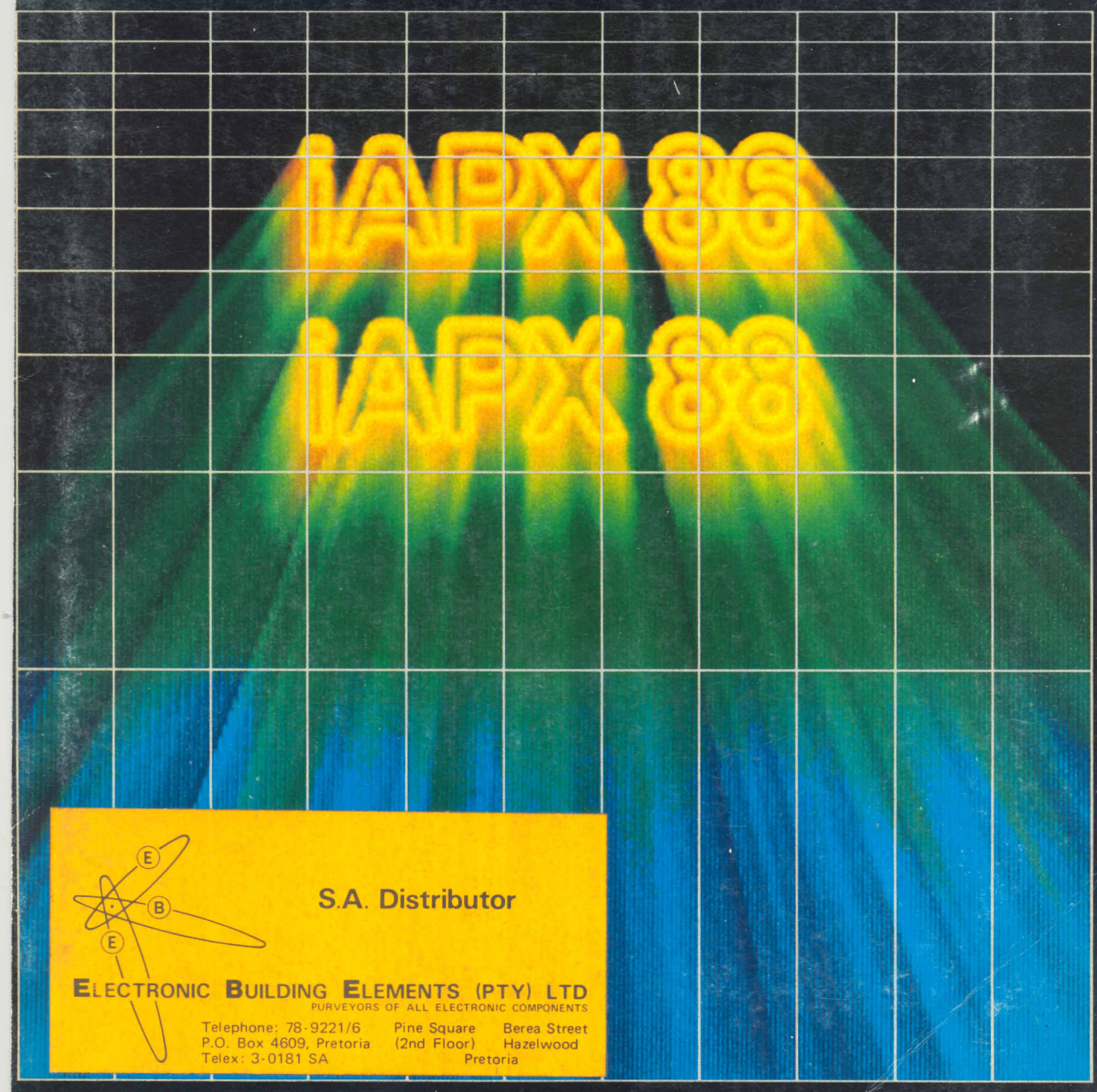
intel

Microsystem 80
iAPX 86 and iAPX 88
Product Description



MICROSYSTEM 80
iAPX 86 AND iAPX 88
PRODUCT DESCRIPTION

JULY 1980

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

AFN-01300A-1

Table of Contents

MICROSYSTEM 80 NOMENCLATURE

The increase in microcomputer system and software complexity has prompted Intel to introduce a new family of microprocessor products to reduce application complexity and cost. This new generation of Intel microprocessors is powerful and flexible and includes many processor enhancements. These include CPUs, numeric floating point extensions, I/O processors, and all the support chips required for a full function system.

As Intel's product line has evolved, its component based product numbering system has become inappropriate for all the possible VLSI computer solutions offered. While the components retain their names, Intel has moved to a new *system-based* naming scheme to accommodate these new VLSI systems.

We have adopted the following prefixes for our product lines, all of them under the general heading of Microsystem 80:

iAPX — Processor Series
iRMX — Operating Systems
iSBC — Single Board Computers
iSBX — MULTIMODULE Boards

Concentrating on the iAPX Series, two processor lines are currently defined:

iAPX 86 — 8086 CPU based system
iAPX 88 — 8088 CPU based system

Configuration options within each iAPX system are identified by adding a suffix, for example:

iAPX 86/10 — CPU Alone (8086)
iAPX 86/11 — CPU + IOP (8086 + 8089)
iAPX 88/20 — CPU with Math Extension (8088, 8087)
iAPX 88/21 — CPU with Math Extension + IOP (8088, 8087 + 8089)

This improved numbering system will enable us to provide you with a more meaningful view of the capabilities of our evolving Microsystem 80.

The Software Crisis

The growth of the electronics industry in the past decade has been phenomenal. The 1980's promise to be equally rewarding, as VLSI makes possible new and more complex applications for microelectronics.

The ability to build increasingly complex devices has been the driving force behind the success of the industry. This growth in complexity will continue in the 1980's. Our task will be to exploit this driving force in a way that will help you profit from this increasing complexity.

The challenge of the 1970's was to reduce the cost of the electronic functions needed to store and process data. Our approach was to integrate more and more *microcomputer hardware* on a silicon chip at continually decreasing prices.

The challenge of the 1980's is different. Now we must reduce the cost of electronic solutions; that is, reduce the cost you incur in using our devices to build products. Solving this problem will require a shift from the component integration that was prevalent in the 1970's to concentration on *system level* integration in the 1980's.

The reason for this shift is a direct result of the success of the 1970's. The declining price of LSI devices made it possible to put microelectronics into everything from toys to automobiles to appliances. At the same time, the capability of the microcomputer was also growing. Used first for simple logic replacement, the microcomputer became powerful enough for dedicated computer applications and then moved on to become a building block for user-programmable computer systems.

We can now talk about putting the power of a mainframe CPU on a single chip. This buys you nothing as a customer, however, unless you can use that power. Hardware is computing *potential;* it must be harnessed and driven by software to be useful.

Calculating the requirements for computer programmers for all the new microcomputer products, we come up with a need for about *one million* programmers by 1990! When we look at the fact that U.S. electronic engineering schools produced only about 17,000 graduates in 1979, the challenge looks even greater. It is clear that our success is leading to what we call the "software crisis".

There is an answer. In the 1980's, we will replicate our past success by integrating *standardized software.* That will help you develop and maintain microcomputer applications less expensively.

The present level of integration is characterized by the integrated central processor such as our 8086: a CPU on a chip. We have developed and are introducing additional iAPX 86,88 support processors, such as a math coprocessor, the 8087, and an input/ output processor, the 8089. Either of the devices, in conjunction with an 8086 or 8088 CPU, can drive an extremely powerful math or I/O oriented system.

In the future, we will extend the process by integrating operating system and software functions into the processing hardware. Engineering and programming costs will be reduced substantially, and you will have an upward compatible interface for future products. Our goal in the 1980's is to deliver the optimum level of system integration for the complexity level of each of your applications. What this will give you is a *minimum total cost* solution for your application.

This point is best illustrated in terms of dollars. In the early 1970's, producing a simple system using components cost a few hundred thousand dollars. Now, the typical cost of system implementation nears a million dollars. The complex system of the 80's will cost perhaps five million dollars to implement, assuming the programmers can be found to do the job! By providing the optimum level of system integration, as described, your costs are reduced dramatically (see Figure 1).

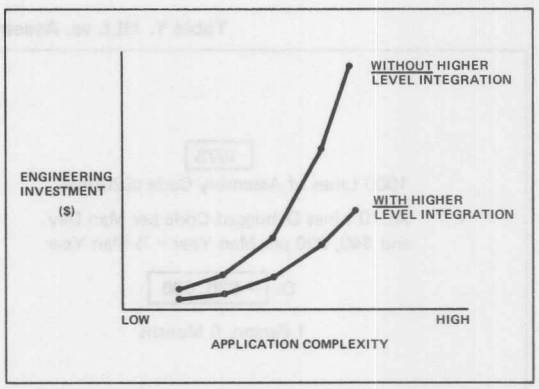


Figure 1. Typical Application Total Engineering Investment

This approach was the rationale for the design of the evolving Microsystem 80 series. You will see that much of our investment in the future of the iAPX 86 and iAPX 88 product lines is aimed at integrating software into the hardware to offer you the most cost effective solution for your application needs in the 1980's.

The iAPX 86,88 provides the foundation for meeting our objectives for the 1980's: to offer you the world's highest performance VLSI systems, and to provide you the total solution at the lowest possible cost.

Evolving Application and Software Complexity

Today's complex microsystem applications range from dedicated 8-bit control processors to multi-tasking 8-bit, 16-bit, and 32-bit systems. Intel plans to provide a product line that will address the full range of these microsystems.

As the complexity of your product line increases — for example, from a small single user data processor to a large multi-user data processor, or from an intelligent terminal to a shared

logic word processor — the level of software complexity increases *more* than proportionally. Application software is rapidly headed in the direction of the full blown "mainframe-style" multiprocessing environment. This application trend is the basis for the "software crisis".

The iAPX 86 and iAPX 88 address this growing software complexity head on. They provide an architecture *designed* for high level languages. High level languages such as PASCAL and FORTRAN are much more cost effective than assembly language for large software applications.

High level languages offer a major productivity improvement in application development. They use simple, algorithmic statements, independent of processor architecture to define the application. This reduces errors and produces programs that are far easier to understand, debug, and maintain than if done in assembly language. HLL programming increases programmer productivity, benefiting you with lower costs and earlier completion. You realize returns on your investment quicker and reduce the total investment necessary.

Table 1 shows a case study comparing costs of assembly language coding and HLL programming. In this study we have used the generally accepted program development rate of about 10 lines of debugged code per day — *regardless of programming language.*

This example shows that pure assembly language coding is no longer a cost effective alternative in large applications. The price of a line of code, whether assembly or HLL, is estimated to be $50-60. HLL programming can produce over four times the output of assembly coding per dollar. The cost implications of the wrong choice in a large, complex application can be enormous.

Table 1. HLL vs. Assembly Language Cost Analysis

1975

1000 Lines of Assembly Code ($20/line)

At 10 Lines Debugged Code per Man Day, and $40, 000 per Man Year = ½ Man Year

Or **≃ $20, 000**

1 Person, 6 Months

1980

13, 000 Lines of Assembly Code ($50.00/line) or
3, 000 Lines of HLL Code ($50.00/line)

At 10 Lines Debugged Code per Day and $125, 000 per Man Year =

Assembly Language	High Level Language
6.5 Man Years	1.5 Man Years
≃ $800, 000	**≃ $180, 000**
3 People, 2+ Years	2 People, 9 Months

Microsystem 80 Introduction

1

iAPX 86 AND iAPX 88 ARCHITECTURE — THE FOUNDATION FOR THE FUTURE

Overview

iAPX 86,88 is an evolving family of microprocessors and peripherals. The family partitions processing functions among general data processors (8086 and 8088), specialized coprocessors like the 8087 numeric data processor, and I/O channel processors (the 8089).

Four key architectural concepts shaped the data processor designs. All four reflect the family's role as vehicles for modular, high level language programming (in addition to assembly language programming). The four architectural concepts are memory segmentation, the operand addressing structure, the operation register set, and the instruction encoding scheme. They are distinct departures from the minicomputer architectural styles of the 1960's and 1970's.

These earlier architectures (minicomputers) were designed for assembly language programming which emphasizes register based data and linear programs. Over the last decade, large software development projects shifted their programming to high level languages which employ modular programming and memory based data. The iAPX 86,88 memory segmentation scheme is intended for modular programs. It supports the static and dynamic memory requirements of program modules, as well as their communication needs. The iAPX 86,88 registers are designed for fast high level language execution. The scheme employs specialized registers and implicit register usage. You will derive significant performance and memory utilization improvements directly from these architectural features.

The four concepts are discussed in the following sections. They are:

- Memory segmentation for modular programming, evolution to memory management and protection
- Addressing structure for high level programming languages
- Operation register set for computation
- Instruction set encoding for memory efficiency and execution speed

Memory Segmentation for Modular Programming

Large programs (10-100K bytes) are not generally written in assembly language. They are developed in individually compiled modules in high level languages. Modular program development techniques, program libraries, compatible linking, and project management tools are often requirements in such an environment. A complex application program might be composed of multiple processes, with each process constructed from multiple modules. Processes send messages to each other for communication, while modules generally share common data when needed. Ideally, these inter-module communication paths are well structured and disciplined.

The iAPX 86,88 segmentation scheme is optimized for the reference patterns of computer programs. Four segment registers are provided in a segment register file. Memory references are relative to automatically selected code segment (CS) and data segment (DS) registers. The module shares a stack segment (SS) with all other modules of the process (task). The module may share a global data segment with other modules in the process; for example, to send and receive messages between modules. This segment is accessed explicitly with the extra segment (ES) register.

This scheme is highly efficient because constant program references to code and data, as well as the stack, have *automatic* segment selection. This results in minimized instruction length. Only 16 bits are required to address anywhere in the full megabyte address range. Only infrequent inter-module communications require the extra prefix bits to explicitly override the automatic segment selection.

There are two other significant advantages to the segment register concept. First, it separates segment base addresses from offset addresses which are relative to the segment base. Only offset addresses are used within object modules. This supports position-independent, dynamically relocatable modules. You merely have to alter the CS and DS register contents to move a module, rather than relinking the whole task and reloading. This structure employs short addresses (16 rather than 20-bit) for efficient use of memory.

The second advantage of iAPX 86,88 segmentation is that it can be extended to include memory management and multi-level protection. The contents and width of segmentation registers are independent of the rest of the instruction set. The architecture can be made to address additional memory and provide access rights and limit checking. Using the mainframe concept of memory based segment tables, this structure can also support virtual memory. Further, since only four registers are active in the file at a time, these features can be accomplished on the CPU chip itself, avoiding the access delays of off-chip memory management.

In summary, memory segmentation has several ultimate benefits for the end user. It provides for simplified hardware and faster, modular software development, more easily maintainable code, and provides an orderly way for the architecture to grow.

Addressing Structure for High Level Programming Languages

The iAPX 86,88 architecture employs an operand addressing scheme complementing the memory segmentation scheme. There are four components in an address. They are the segment, base, index, and displacement. The segment component was just described. A base register is dedicated to both the data and stack segments. These base registers may

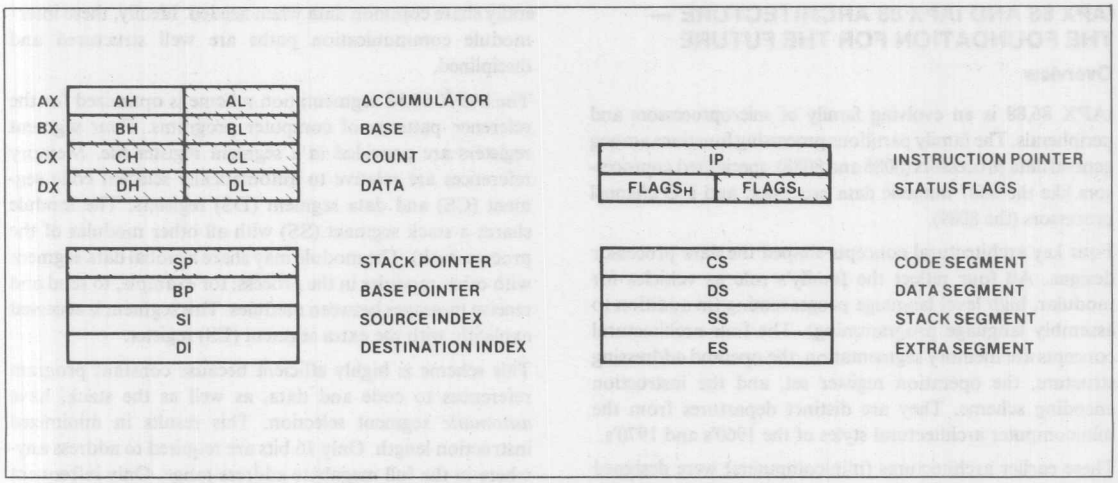


Figure 2. iAPX 86/10, 88/10 Register Model

also be used when accessing the extra (global) segment. They are used for holding the base address of a data structure.

Two index registers are provided for use with the base registers to dynamically select any element from a based data structure. Eight or sixteen-bit fixed displacements may be added to any of these address forms. The complete register file is shown in Figure 2 and the addressing structure is shown in Figure 3.

Referring to Figure 3, an iAPX 86,88 operand address contains up to four components: a segment (S), a base (B), an index (I), and a displacement (d). The segment component is automatically selected for the code, data, and stack segments. An explicit segment selection is required for data references in the extra segment. Any combination of the remaining three address components is permitted in virtually all memory reference instructions, with at least one always being present.

Block and string data are extensions to this scheme. They use different assumptions for source and destination segments, but the segments are still implicitly accessed. Immediate operands are also supported.

The iAPX 86,88 is a two operand machine (source and destination). It supports source/destination operand combinations of register/memory, memory/register, memory/memory (string operations only), immediate/register, and immediate/memory. The various address combinations of S, B, I, and d correspond to common data structures used in high level language programming. Such data structures can therefore be implemented easily in assembly language as well.

Figure 3 shows the correspondence between the most common iAPX 86,88 address modes and various data types in high level programming languages. The S component is

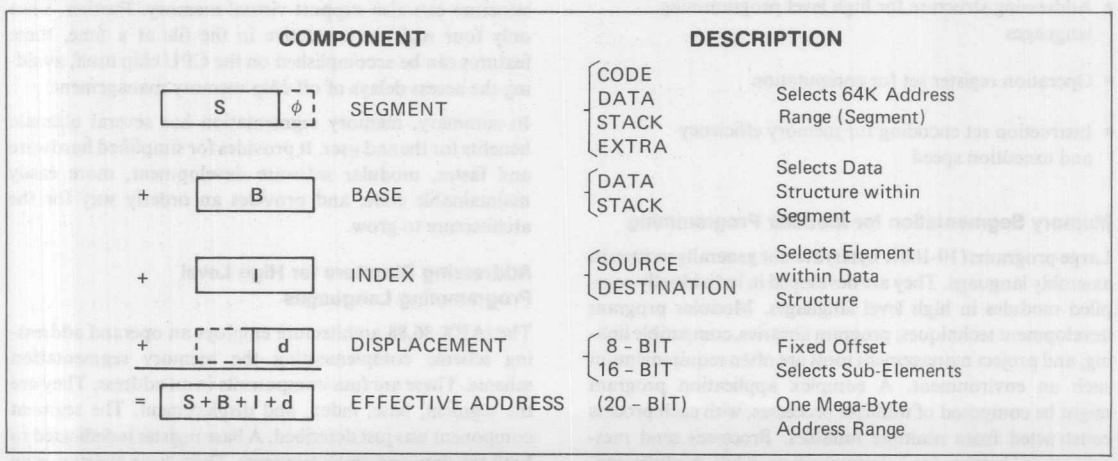


Figure 3. iAPX 86,88 Four Component Addressing Structure

implicit; the stack base (BP) assumes the stack segment; no B component, or use of the data base (BX), assumes the data segment. The less commonly used address modes are not shown.

The stack base (BP) is a concept borrowed from the family of P-machines "developed" as ideal PASCAL vehicles. P-machines term this register the "mark pointer". It always points to the base of the current local data area in the stack segment. This permits efficient local addressing in block-structured languages such as PASCAL and PL/M. In these languages, procedures are invoked by pushing their parameters on the stack, calling the procedure, and then allocating their local data area on the stack. The iAPX 86,88 return instruction then removes the parameters from the stack, as is done in the P-machines.

Operation Register Set for Computation

The Intel iAPX 86,88 line is truly a complete family of microprocessors. The iAPX 86/10 and iAPX 88/10 are the general data processor members of the family, while the 8089 is the I/O processor family member. In addition, the CPU itself has an interface for attaching coprocessors. Coprocessors provide specialized operation set extensions that benefit the application by performing special purpose logic to increase performance.

The iAPX 86/20 Numeric Data Processor is an example of this concept. Using an 8086 with an 8087 coprocessor (CPU extension) it provides a *one hundred-fold performance boost* over the iAPX 86/10 for a wide range of numeric operations. The full computational capability of the iAPX 86,88 family can therefore span a much broader range than is possible with a single microprocessor. This technique has been used successfully in the mainframe and minicomputer industries to provide instruction set options for scientific, commercial, text processing, or other special purpose applications.

An 8087 extends the iAPX 86 or iAPX 88 architecture to include additional data types, registers, and instructions. The 8086 or 8088, with an 8087 coprocessor, operates on 16, 32, and 64-bit integers, 32, 64 and 80-bit floating point numbers, and up to 18 digit packed BCD numbers. Data conversions and calculations are performed in the 8087 and are transparent to the programmer.

The iAPX 86/10 and iAPX 88/10 CPUs alone can perform arithmetic operations on signed and unsigned 8 and 16-bit binary integers as well as packed and unpacked decimal integers. The full complement of logical operations are provided as well. Interesting new features are the string operations. Six primitive string instructions (move, skip, search, compare, set, and translate) are standard. When combined with special control operators, complex string manipulations are possible with two or three *instructions.*

Instruction Set Encoding for Memory Efficiency and Execution Speed

The iAPX 86 uses a byte oriented instruction stream while operating with a 16-bit data bus. To accomplish this, the processor is subdivided into two independent parallel processors called the bus interface unit (BIU) and the execution unit (EU). The iAPX 88 employs an identical execution unit and is 100% code compatible with iAPX 86, yet it interfaces to an 8-bit wide data bus BIU. The bus interface unit is an independent processor that prefetches instructions. Instruction fetch time is therefore mostly overlapped with other iAPX 86,88 processor activity. The bus interface unit permits either instructions or data to be placed in memory without regard to word boundaries. (An array of five byte records in PASCAL can be referenced without requiring an additional byte of padding to word align the records.) Processor subdivision into the BIU and EU has the additional benefit of minimizing the effect of wait states and bus hold time on CPU efficiency.

Instruction set encoding is substantially improved when instructions are composed in byte multiples instead of words. Instructions in the iAPX 86,88 vary from one to six bytes in length (not counting optional prefix bytes). The average instruction is three bytes long. In a word aligned machine the same information would occupy four bytes. This and the features described above give the iAPX 86,88 roughly a 30% program space savings over other architectures.

PROCESSOR PARTITIONING

Beyond efficient support for high level languages, the iAPX 86 and iAPX 88 establish the foundation for the family to build on in the 1980's. The family uses increasing levels of integration to significantly reduce software, hardware, and development investment.

The iAPX 86/10 and iAPX 88/10 general purpose processors employ external module integration. Specialized system functions are distributed among optimized components and removed from the host processor. The CPU is freed to become the system manager and resource allocator rather than doing "all things for all programs". The family also includes the 8087 Numeric Data Processor and the 8089 I/O Channel Processor.

These processors are optimized to address the three main functions in a computer environment: data processing and control, arithmetic computation, and input/output. The 8087 and 8089 are described below.

The 8087 Numeric Processor Extension (NPX) adds over 50 numeric opcodes and eight 80-bit registers to the host processor to provide more extensive data and numeric processing capability. It performs floating point and trans-

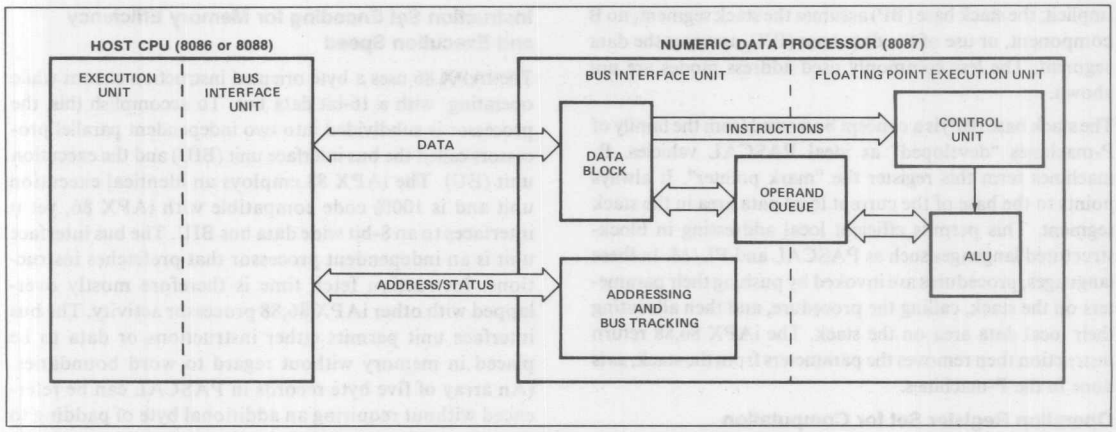


Figure 4. Numeric Data Processor Block Diagram

cendental (trigonometric) functions, processes decimal operands up to 18 digits without roundoff, and performs exact arithmetic on integers up to 64 bits long. Another feature of the NDP, with important benefits to you, is that it is compatible with the proposed IEEE floating point standards. It can be used in applications requiring high speed computation such as numerical analysis, accounting and financial applications, the sciences, and engineering. Throughput increases in such applications up to *100 times* current speeds are typical (See Figure 4.)

The 8089 Input/Output Processor (IOP) is an independent microprocessor that optimizes input/output operations. The objective of the IOP is to remove all I/O details from application software. It responds to CPU direction but executes its own instruction stream in parallel with other processors. I/O transfers of either 8 or 16-bit data can be done at rates up to 1.25 megabytes per second. The IOP therefore combines the attributes of both a CPU and a DMA controller to provide a powerful I/O subsystem. An important feature of the IOP is that it can be physically isolated from the application CPU. The advantage to you is that I/O subsystem changes or upgrades can be made without any impact to application software. (See Figure 5.)

Summarizing, there are several advantages to external module integration:

- System tasks may be allocated to special purpose processors designed for optimal task handling
- Simultaneous operation (parallel processing) provides highest system performance
- Isolated system functions minimize the effect of modifications, local failures, or errors on the rest of the system

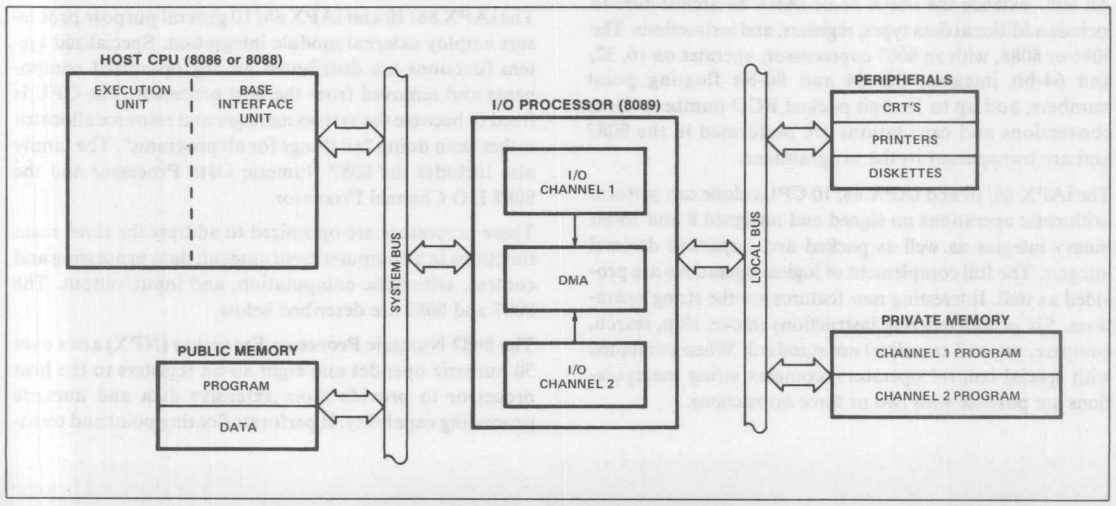


Figure 5. I/O Processor Block Diagram

- The iAPX 86,88 family of processors allows division of the application into small, manageable tasks for parallel development, while providing built-in hardware facilities for coordinating processor interaction. With the iAPX 86,88 approach you can implement high performance systems far more quickly and easily than would otherwise be possible.

DEVELOPMENT TOOLS

Development Systems

Development systems are a unique combination of hardware and software tools which increase your product development productivity. With Intel development products, you will shorten the development cycle and reduce your time to market.

Development systems from Intel provide an upgradable spectrum of tools ranging from stand alone development systems to future networks of specialized work stations. Intel eliminates your risk of development system obsolescence by guaranteeing product upgradability and compatibility. This guarantee protects your capital investment.

For small to medium size projects, the Intellec™ development system is available in many configurations at low cost. For small projects, these systems have nominal program memory with floppy disks as peripheral storage devices. Minimum configurations may be upgraded to provide increased performance, increased memory, and increased mass storage via hard disk. These more powerful configurations support medium sized projects.

The Intellec Series II/85 is a good example of such a system. It is a complete microcomputer development system integrated into one compact package. The Model 225 includes a CPU with 64K bytes of RAM, 4K bytes of ROM, a 2000 character CRT, detachable full ASCII keyboard, and a 250K byte floppy disk drive. The powerful ISIS-II Disk Operating System software allows you to efficiently develop and debug iAPX 86,88 programs. Optional storage peripherals provide over 2 million and 7.3 million bytes of storage on floppy and hard disk, respectively.

Distributed development configurations address the range of medium to large sized projects. These configurations connect multiple standalone development systems to more powerful support resources such as mainframes and their peripherals.

In addition to the Intellec® development system, Intel offers several products to help you debug and test your hardware and software. In-Circuit-Emulators, such as ICE-86™ and ICE-88™, are available to emulate your product environment. They increase development productivity substantially. Another software tool, RBF-89, helps you debug 8089 software under ICE control. With these tools, software development time can be reduced dramatically — lowering your total investment.

High Level Languages

Programming languages are the key to developing an application. Intel programming languages serve three purposes in your design. First, they are your primary design tool. Intel's breadth of languages and extended features give you the maximum ability to properly design and plan your program. Second, Intel languages are a communication vehicle between programmers during implementation and later during modification. Standard high level languages allow programmers to better communicate what the programs do. Third, Intel languages are designed in conjunction with Intel microsystems to provide the greatest code efficiency and execution speed. Intel languages speed implementation of your design and reduce maintenance costs.

MDS-311 is a set of software development tools for iAPX 86 and iAPX 88 applications. It is a complete set of software products that run on the Intellec Model-800 and Series-II development systems. The software tools provided include PL/M-86, high level programming language, and the ASM-86 assembler. Two utilities, LINK86 and LOC86, are supplied to link separately compiled or assembled program modules into executable tasks. The Library Manager, LIB86, lets you maintain a library of iAPX 86 or iAPX 88 object modules. These modules can then be linked in with new programs without being recompiled. This simplifies and speeds your development. Common code (e.g. a subroutine) only has to be developed and compiled once. Intel code converters, such as CONV86, are very useful tools for migrating 8080 or 8085, Z80, and 6809 assembly language programs to the iAPX 86 or iAPX 88. They convert assembly source code to ASM86 source code. This will help you make a rapid transition and cut redevelopment costs substantially.

Intel will provide a variety of languages for both systems and applications to facilitate development of your product. You can choose the language (or languages) which best suits your product needs and the expertise of your staff. ASM86, the assembly language, and PL/M-86, the systems oriented high level language, are both currently available. PASCAL, FORTRAN, and BASIC will be offered in the near future, and COBOL is planned after that.

Intel's languages also run on your final product. Your product's function is significantly increased when packaged with language translators. They allow your customers to tailor your products for their environment. Intel's languages will save implementation time and free resources to work on the value-added portion of your product.

SINGLE BOARD COMPUTERS ACCELERATE YOUR MICROSYSTEM SUCCESS

In addition to the increased integration of functions in VLSI components, there is a strong trend today to implement microsystem applications with single board compu-

ters. This allows the design engineer to:

- Easily configure reliable and cost-effective systems using iSBC and iSBX standard products.
- Overcome the shortage of qualified engineers and technicians.
- Get the end product to market quickly.
- Focus on the application.
- Offset the increasing cost of capital.

In addition, using iSBC single board computers and iSBX expansion products in your design reduces the number of risks that you must face in all phases of the product life cycle. The four major risk areas that Intel iSBC and iSBX products will help you overcome are as follows:

1. Limited Resources

Using a fully tested board computer, which incorporates the key elements of processor, memory and I/O, helps overcome today's critical shortage of engineers, programmers and technicans. Implementing iSBC boards and iSBX MULTIMODULES in your design reduces increasing capital costs in production, QC, and test. It is estimated that using iSBC boards can save up to $200,000 per board design.

2. Time to Market Dictates Success or Failure

With inflation running at its current rate, the amount of time it takes to get a product from an idea to the market becomes critical. A delay of a few months can collapse your return on investment.

Experience shows that the first company that gets its product to the marketplace usually dominates that market. You can get your product to the market months earlier using standard off-the-shelf iSBC, iSBX and Real-Time Executive (iRMX) Software modules. Intel's large board manufacturing and distribution capability enables you to respond to your market demand rapidly and in a cost-effective manner.

3. Solution Completeness and Project Credibility

Microprocessor based solutions for today's problems are commonplace and are expected to succeed. A broad spectrum of compatible system components in the iSBC, iSBX, and iRMX product line increase the probability of being right the first time. General purpose iSBC board solutions are easy to customize through the use of iSBX modules from Intel, or your own design.

4. Coping with the Technology/Complexity Avalanche

iSBC and iSBX products incorporate the latest in VLSI shortly after their initial introduction. With increasing system complexity Intel's design process and testing reduces the risk of "gremlin" bugs which multiply with complexity and evade diagnosis. Standards used throughout the product family such as the de facto industry standard MULTIBUS, EIA, IEEE etc. provide a smooth transition for your product to new and changing processor, memory and I/O technologies.

Intel's single board computer product family is continuing to reduce your risk and protect your investment in the future by expanding iSBC and iSBX products in three dimensions: processors, memory, and I/O.

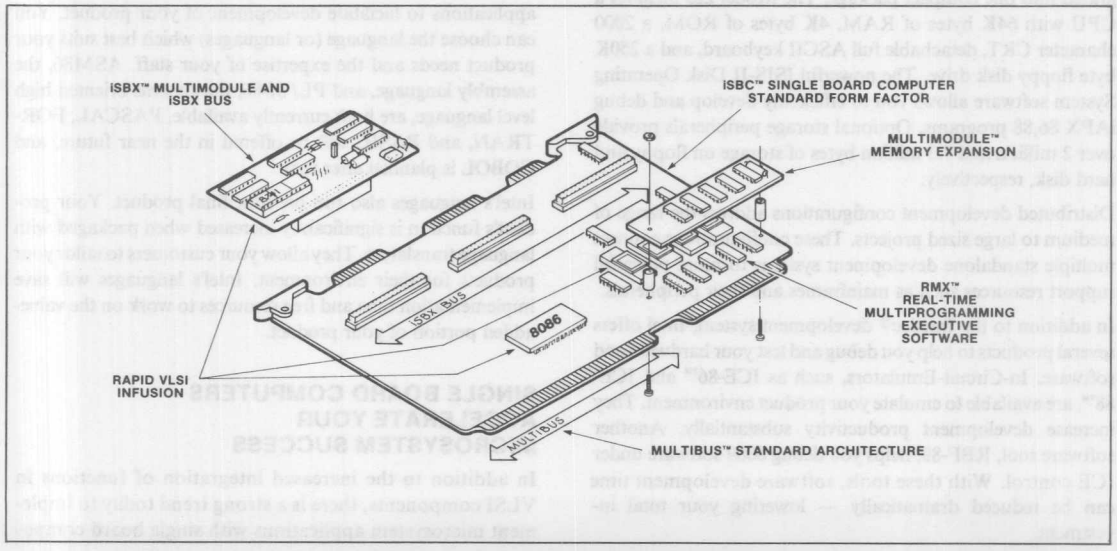


Figure 6. Single Board Computer (iSBC 86/12™)

SUMMARY

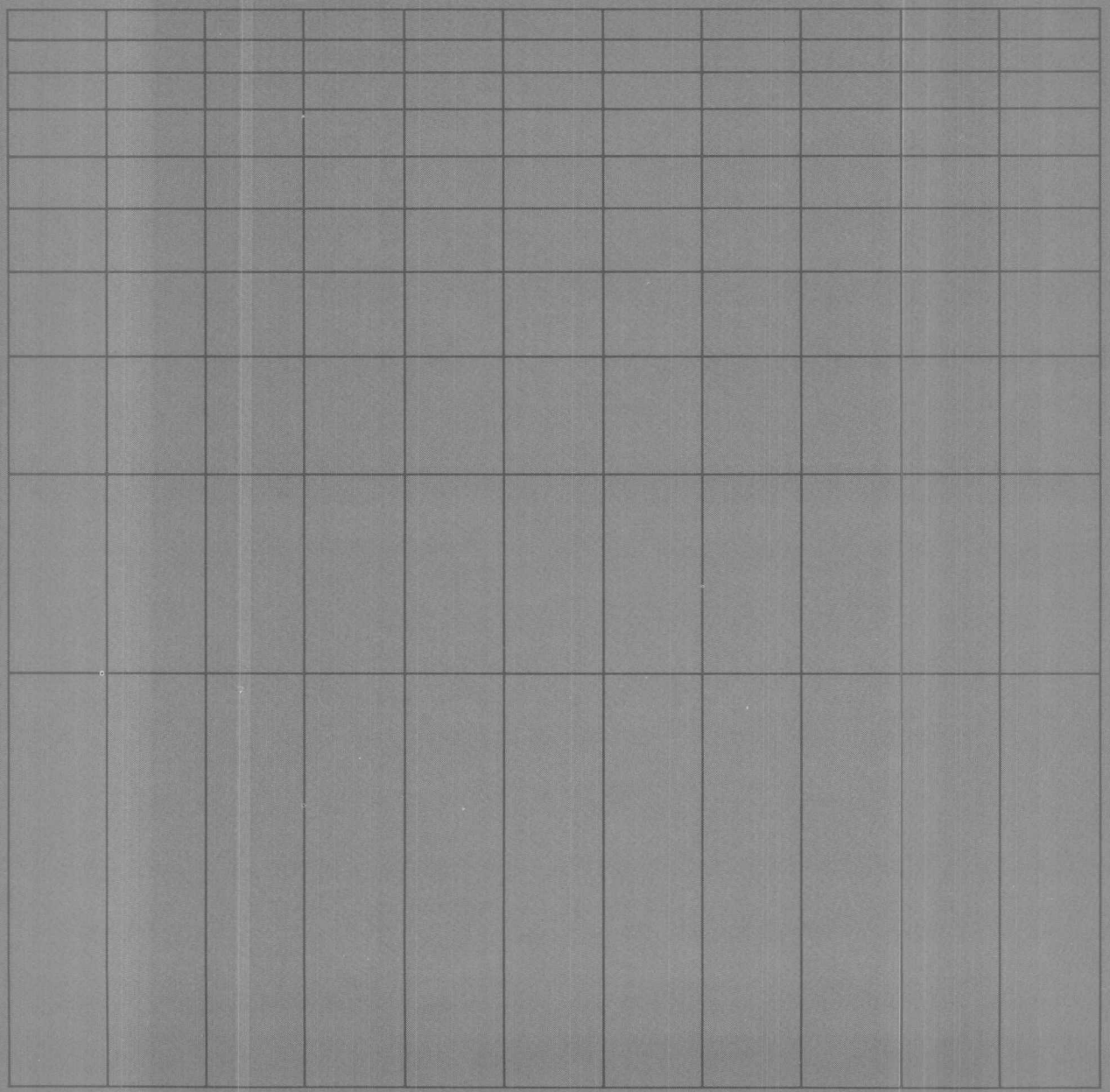
Intel's iAPX 86,88 multiple processor family is designed for modular programming in high level as well as assembly languages.

- Its memory segmentation scheme is optimized for the reference needs of computer programs, and is separate from the operand addressing structure.
- The structure for addressing operands within segments directly supports the various data types found in high level programming languages.
- The family provides an operation register set to support general computation requirements. It also provides for optimized operation register sets to do specialized data processing functions with its inherent multi- and coprocessor support.
- The family uses optimized instruction encoding for high performance and memory efficiency
- The family is well supported with development tools and single board computer products.

This architecture provides the foundation for solving the application needs in the 1980's. It makes a noted departure from architectures of the 1960's and 1970's — based on Intel's intent to minimize software and hardware product costs for you, the end user.

Microsystem 80 Component Families

2



iAPX 86, 88

- **Complete high performance microprocessor systems**
- **Multiple independent processing modules**
- **Flexible, intelligent I/O channel functions**
- **Multiple asynchronous bus structures supported**
- **MULTIBUS™ system compatible interface**
- **24 Operand addressing modes, 1 megabyte addressability**
- **Implements the proposed IEEE floating point standard**
- **14 general data processing registers plus an 8x80 bit numeric register stack**
- **Bit, byte, word, real, decimal and block operations**

The components in the iAPX 86, 88 product lines have been designed to operate together in diverse combinations within the systematic framework of the overall family architecture, as shown in Figure 1. In this way a single family of components can be used to solve a wide array of microcomputing problems. A component mix can be tailored to fit the performance needs of an application precisely, without having to pay for unneeded capabilities that may be bundled into more monolithic, CPU-centered architectures. Using the same family of components across multiple systems limits the learning curve problem and builds on past experience. Finally, the modular structure of the family architecture provides an orderly way for systems to grow and change.

The iAPX 86, 88 product line architecture is characterized by three major principles:

1. System functions are distributed among specialized components.
2. Multiprocessing capabilities are inherent in the hardware.
3. A hierarchical bus organization provides for the complex data flows required by high-performance systems without burdening simpler systems with unneeded capabilities.

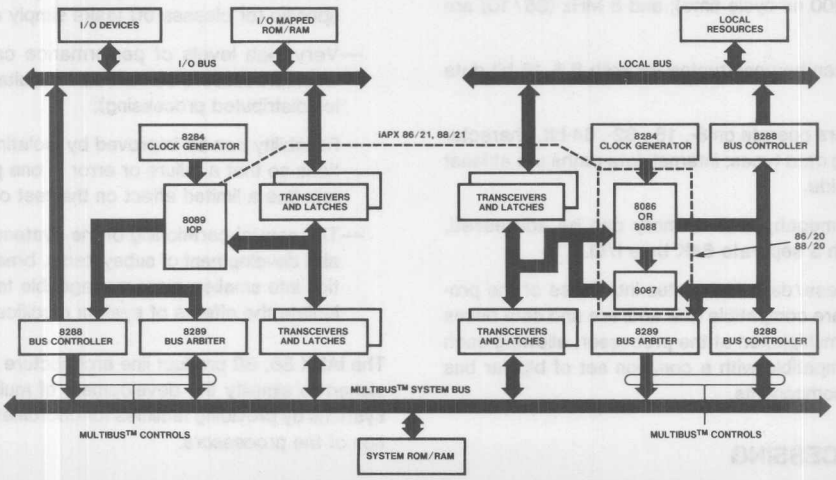


Figure 1. iAPX 86, 88 Multiprocessing System

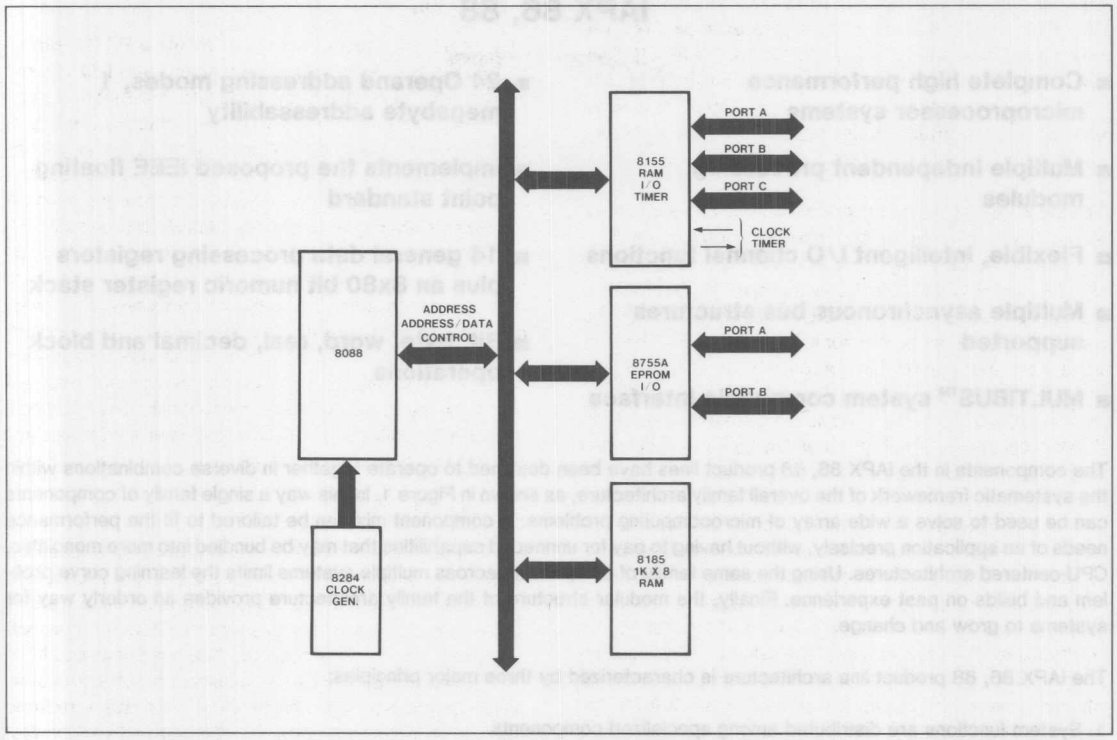


Figure 2. iAPX 88 Allows For A Highly Integrated, High Performance System Using 8085 Family of Components

MICROPROCESSORS

At the core of the product line are four microprocessors that share these characteristics:

—5 MHz (200 ns cycle time); and 8 MHz (86/10) are available.

—System can be constructed for both 8 & 16 bit data paths

—Processors operate on 8-, 16-, 32-, 64-bit, character and string data types; internal data paths are at least 16-bits wide.

—Up to 1 megabyte of memory can be addressed, along with a separate 64K byte I/O.

—The address/data and status interfaces of the processors are compatible (the address and data buses are time multiplexed at the processor, allowing each to be compatible with a common set of bipolar bus support components.

MULTIPROCESSING

Employing multiple processors in medium to large systems offers several significant advantages over the centralized approach that relies on a single CPU and extremely fast memory:

—System tasks may be allocated to special-purpose processors whose designs are optimized to perform specific (or classes of) tasks simply and efficiently;

—Very high levels of performance can be attained when processors can execute simultaneously (parallel/distributed processing);

—Reliability can be improved by isolating system functions so that a failure or error in one part of the system has a limited effect on the rest of the system;

—The natural partitioning of the system promotes parallel development of subsystems, breaks the application into smaller, more manageable tasks, and helps isolate the effects of system modifications.

The iAPX 86, 88 product line architecture is explicitly designed to simplify the development of multiple processor systems by providing facilities for coordinating the interaction of the processors.

The architecture supports two types of processors: independent processors and coprocessors. An independent

processor is one that executes its own instruction stream. The iAPX 86/10, 88/10, and IOP are examples of independent processors. An iAPX 86/10 or iAPX 88/10 typically executes a program in response to an interrupt. The IOP starts its channels in reponse to an interrupt-like signal called a channel attention; this signal is typically issued by a CPU.

The iAPX 86, 88 product line architecture also supports a second type of processor, called a coprocessor. Coprocessor "hooks" have been designed into the iAPX 86/10 and iAPX 88/10 to allow this type of processor to be accommodated in the future. A coprocessor differs from an independent processor in that it obtains its instructions from the independent processor or host. The coprocessor monitors instructions fetched by the host and recognizes certain of these as its own and executes them. A coprocessor, in effect, extends the instruction set (and architecture) of its host processor.

The iAPX 86, 88 product line architecture provides built-in solutions to two classic multiprocessing coordination problems; bus arbitration and mutual exclusion. Bus arbitration may be performed by the bus request/grant logic contained in each of the processors (local bus arbitration), by 8289 bus arbiters (system bus arbitration), or by a combination of the two when processors have access to multiple shared buses. In all cases, the arbitration mechanism operates invisibly to software.

For mutual exclusion, each processor has a LOCK (bus lock) signal which a program may activate to prevent other processors from obtaining a shared system bus. The IOP may lock the bus during a DMA transfer to ensure both that the transfer completes in the shortest possible time and that another processor does not access the target of the transfer (e.g., a buffer) while it is being updated. Each of the processors has an instruction that examines and updates a memory byte with the bus locked. This instruction can be used to implement a semaphore mechanism for controlling the access of multiple processors to shared resources. (A semaphore is a variable that indicates whether a resource, such as a buffer or a pointer, is "available" or "in use".

BUS ORGANIZATION

Figure 3 summarizes the iAPX 86, 88 bus structure. There are two different types of buses: system and local. Both buses may be shared by multiple processors, i.e., both are multimaster buses. Microprocessors are always connected to a local bus, and memory and I/O components usually reside on a system bus. The iAPX 86, 88 bus interface components link a local bus to a system bus.

Local Bus

The local bus is optimized for use by the iAPX 86, 88 microprocessors. Since standard memory and I/O components are not attached to the local bus, information can be multiplexed and encoded to make very efficient use of processor pins (certain MCS-85 peripheral components can be directly connected to the local bus). This allows several pins to be dedicated to coordinating the activity of multiple processors sharing the local bus. Multiple processors connected to the same local bus are said to be local to each other; processors on different local buses are said to be remote to each other, or configured remotely. Both independent processors and coprocessors may share a local bus; on-chip arbitration logic determines which processor drives the bus. Because the processors on the local bus share the same bus interface components, the local configuration of multiple processors provides a compact and inexpensive multiprocessing system.

System Bus

A full implementation of an iAPX 86, 88 system bus consists of the following five sets of signals: address bus, data bus, control lines, interrupt lines and arbitration lines. A group of bus interface components transforms the signals of a local bus into a system bus. The number of bus interface components required to generate a system bus depends on the size and complexity of the system; reduced application needs translate directly into reduced component counts. These main variables determine the configuration of a bus interface group: address space size (number of latches), data bus width (number of transceivers), and arbitration needs (presence of a bus arbiter).

The iAPX 86, 88 system bus is functionally and electrically compatible with the MULTIBUS multimaster system bus used in Intel's iSBC line of single board computing products. This compatibility gives system designers access to a wide variety of computer, memory, communications and other modules that may be incorporated into products, used for evaluation or for test vehicles.

Processing Modules and Bus Topology

The processor(s) and bus interface group(s) that are connected by a local bus constitute a processing module. A simple processing module could consist of a single CPU and one bus interface group. A more complex module would contain multiple processors, such as two IOPs, a CPU and one or two IOPs, or a CPU with a coprocessor with/without IOP. One bus interface group typically links the processors in the module to a public system bus. If there are multiple processing modules in the system, all memory or I/O connected to the public bus is accessible

to all processing modules on the public bus. 8289 bus arbiters in each processing module control the access of the modules to the public bus and hence to the public memory and I/O.

A second bus interface group may be connected to a processing module's local bus, generating a demultiplexed bus. This bus can provide the processing module with a private address space that is not accessible to other processing modules. Distributing memory and I/O resources in this manner can improve system reliability by isolating the effects of failures. It can also increase system throughput dramatically. If processor programs and local data are placed in private memory, contention for use of the public system bus can be held to a minimum to ensure that shared resources are quickly available when they are needed. In addition, processors in separate modules can simultaneously fetch instructions from private memory spaces to allow multiple system tasks to proceed in parallel.

iAPX 86/10, 88/10

- Direct memory addressing capability to 1 MByte
- 5 or 8 MHz clock rate
- 24 Operand addressing modes
- Bit, byte, word, and block operations
- Memory symmetric, segmented architecture for direct high level language support
- MULTIBUS™ system compatible interface
- Multiprocessor, coprocessor and I/O processor extensions supported

The iAPX 86/10, 88/10 are microprocessors whose resources are tailored for efficiency in high level language environments. Most high-level languages store variables in memory; the 86/10, 88/10 symmetrical instruction set

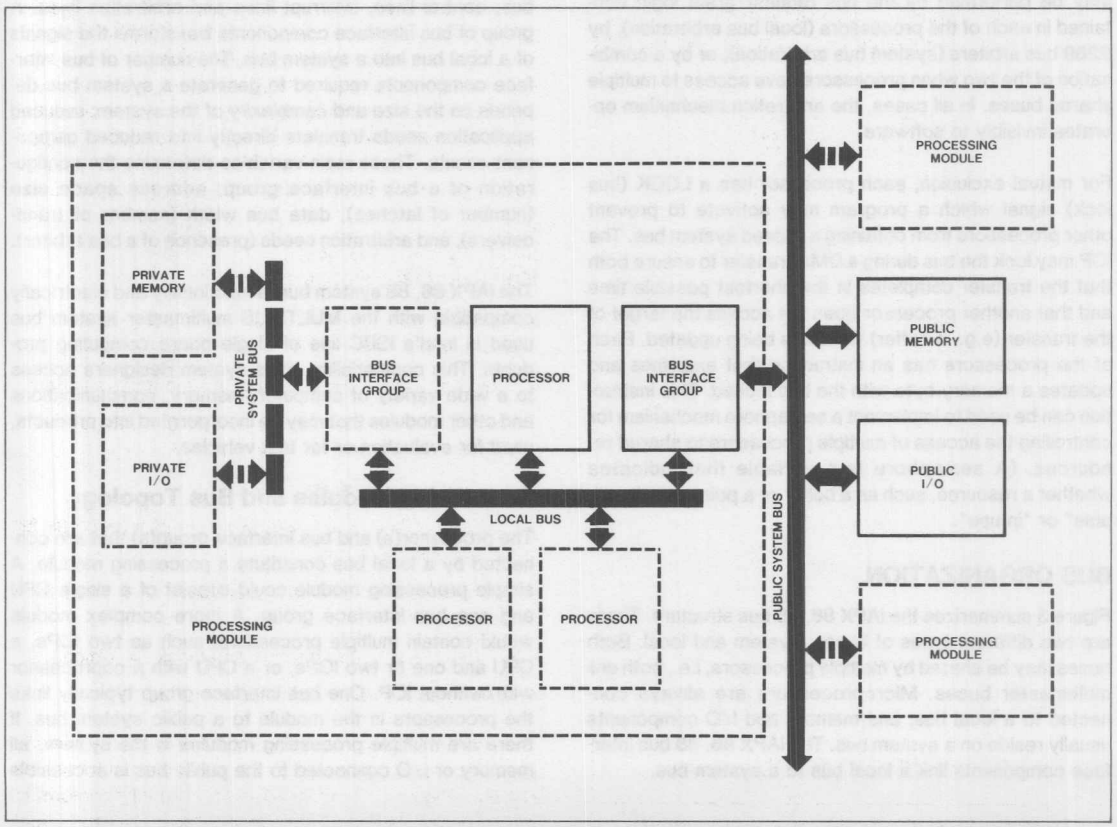


Figure 3. Generalized iAPX 86, 88 Bus Structure

supports direct operation on memory operands, including operands on the stack. The hardware addressing modes provide efficient, straightforward implementations of based variables, arrays, arrays of structures and other high-level language data constructs. A powerful set of memory-to-memory string operation is available for efficient character data manipulation. Of course, routines with critical performance requirements that cannot be met with high level languages may be written in ASM-86 (the 86/10, 88/10 assembly language) and linked with code.

The 86/10 and 88/10 support for high-level languages does not come at the expense of efficient coding. The 86/10, 88/10 instruction set provides byte granularity for all instructions and many optimized versions of more general instructions that are selected automatically by Intel's compiler's and assembler to produce the most compact code possible.

The base architecture of the CPU's themselves allows for extensibility. The CPU's small context makes it especially attractive for larger multiuser, multitasking systems. The 86/10, 88/10 segmented address space and explicit segment control makes it possible to extend the architecture easily to the memory protected environment required in these multiuser systems as well. Multilevel memory protection and a full virtual memory structure can be achieved by interpreting the explicit segment pointers of the 86/10, 88/10 implementation as logical sectors within a virtual address space described by memory resident tables that contain base, limit, and access rights information about each virtual segment. In this way application code written for the 86/10, 88/10 can be transferred directly to the protected environment.

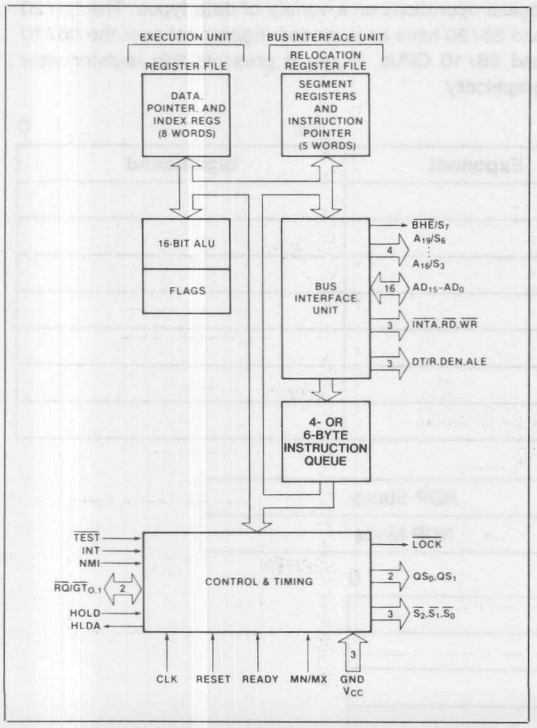


Figure 4. iAPX 86/10, 88/10 CPU Functional Block Diagram

The large application domain of the 86/10 and 88/10 is made possible primarily by the processors' dual operating modes (minimum and maximum mode) and built-in multiprocessing features. Several of the 40 CPU pins have dual functions that are selected by a strapping pin. Configured in minimum mode, these pins transfer control signals directly to memory and input/output devices. In maximum mode these same pins take on different functions that are helpful in medium to large systems, especially systems with multiple processors. The control functions assigned to these pins in minimum mode are assumed by a support chip, the 8288 bus controller.

The high performance of the 86/10 and 88/10 is realized by combining a 16-bit internal data path with a pipelined architecture that allows instructions to be prefetched during spare bus cycles. This makes maximum use of memory bandwidth at the same time actual memory speed (and cost) requirements are relaxed. Also contributing to performance is a compact instruction format that enables more instructions to be fetched in a given amount of time.

iAPX 86/20, 88/20 Numeric Data Processors

- Standard 86/10, 88/10 instruction set plus over 50 new numeric instructions
- Implements the proposed IEEE floating point standard
- High performance, up to 60,000 floating point instructions per second
- Automatic mixed mode arithmetic
- 14 general data processing registers plus an 8×80 bit numeric register stack
- 24 addressing modes available, one megabyte addressability
- Bit, byte, word, real, decimal and block operations
- MULTIBUS™ system compatible interface

The Intel 86/20 and 88/20 are high performance numeric data processors packaged in two 40-pin packages as shown in Figure 1. The 86/20 and 88/20 fully conform to the proposed IEEE floating point standard. The 86/20 and 88/20 provide 68 numeric instructions beyond those

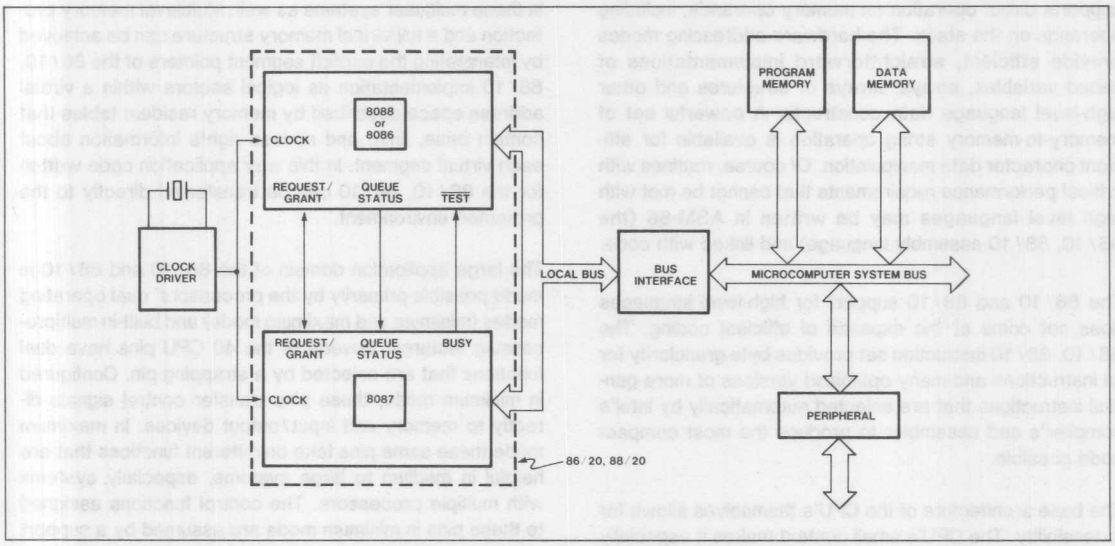


Figure 5. 86/20, 88/20

provided by the iAPX 86/10 and iAPX 88/10, providing a complete solution to high performance numeric and data processing.

PROCESSOR OVERVIEW

The numeric data processor performs arithmetic and logical operations on a variety of data types. The 86/20 and 88/20 have an extended register set over the 86/10 and 88/10 CPUs. Figure 6 presents this register view graphically.

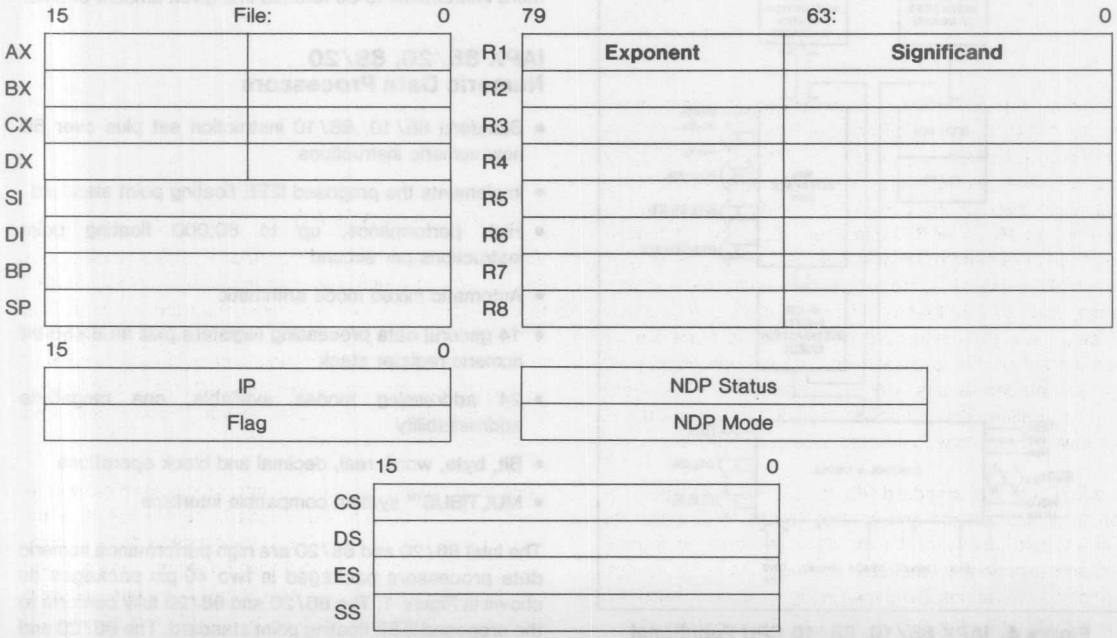


Figure 6. 86/20, 88/20 Architecture

The 86/20, 88/20 support 13 data types. These include six standard types and also seven numeric types, as shown in Figures 7A and 7B.

Format	Length
Byte	8 bits
Word	16 bits
Pointer	32 bits
ASCII	1 char/byte
Packed BCD	2 digits/byte
String	1-64K bytes

Figure 7A. Standard Data Types

DATA FORMATS	RANGE	PRECISION	MOST SIGNIFICANT BYTE
			7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
Word integer	10^4	16 BITS	I_{15} I_0 TWO'S COMPLEMENT
Short integer	10^9	32 BITS	I_{31} I_0 TWO'S COMPLEMENT
Long integer	10^{18}	64 BITS	I_{63} I_0 TWO'S COMPLEMENT
Packed BCD	10^{18}	18 DIGITS	S — D_{17} D_{16} D_1 D_0
Short real	10 ± 38	24 BITS	S E_7 E_0 F_1 F_{23} F_0 IMPLICIT
Long real	10 ± 308	53 BITS	S E_{10} E_0 F_1 F_{52} F_0 IMPLICIT
Temporary real	10 ± 4932	64 BITS	S E_{14} E_0 F_0 F_{63}

INTEGER: I

PACKED BCD: $(-1)^S(D_{17}\ldots D_0)$

REAL: $(-1)^S(2^{E-BIAS})(F_0.F_1\ldots)$

BIAS=127 FOR SHORT REAL

1023 FOR LONG REAL

16383 FOR TEMP REAL

Figure 7B. Numeric Data Types

Internally, the NDP holds all numeric data in a temporary real format (80 bits). This format has extended range and precision. There are key contributors to the NDP's ability to consistently deliver stable, expected results. NDP load and store instructions convert operands represented in memory as 16, 32 or 64 bit integers, 32 or 64 bit floating point numbers (with the range of $+/- 10^{4932}$) or 18 digit packed BCD numbers into temporary real format and vice versa. The fact that these conversions are made, and that calculations are performed on converted numbers, is transparent to the programmer. Integer operands yield correct integer results, decimal operands yield correct decimal results. You control errors such as round off, underflow and overflow in intermediate calculations.

In addition to the standard 86/10, 88/10 instruction set for data manipulation and control, the NDP supports 68 numeric instructions for floating point, trignometric, logrithmic and exponential functions. Example times for several numerical operations are shown in Figure 8. Overall 86/20 system performance is typically 100–130K Whetstones/sec (depending on compiler efficiency), over 50 times the performance of an 86/10 class processor for this scientific benchmark.

APPLICATION AREAS

Viewed strictly from the standpoint of raw speed, the 86/20, 88/20 enable serious computation-intensive tasks to be performed by microprocessors for the first time. The 86/20 and 88/20 provide more than just numeric performance, however. By combining advances made by numerical analysts in the past several years, the NDP's provide a level of usability that surpasses existing minicomputer and mainframe arithmetic units.

Beyond traditional numerics support for scientific applications, the 86/20 and 88/20 have built-in facilities for "commercial" computing. They can process decimal numbers of up to 18 digits without roundoff errors, and perform exact arithmetic on integers as large as 64 bits, allowing many accounting and financial programs to utilize 86/20 or 88/20 performance.

Instruction	Approximate Execution Time (μs)	
	86/20 (5 MHz Clock)	86/10 Emulation of 86/20 (5 MHz Clock)
Add/Subtract Magnitude	14/18	1,600
Multiply (Single Precision)	18	1,600
Multiply (Double Precision)	27	2,100
Divide	39	3,200
Compare	10	1,300
Load (Single Precision)	9	1,700
Store (Single Precision)	17	1,200
Square Root	36	19,600
Tangent	110	13,000
Exponentiation	130	17,100

Figure 8. Execution Time for Selected 86/20 Actual and Emulated Numeric Instructions

PROGRAMMING INTERFACE

Numeric programs for NDP's can be written in ASM-86, the 86/20, 88/20 common assembly language. ASM-86 provides directives for defining all numeric data types and mnemonics for all instructions. All 86/10, 88/10 addressing modes are available in the 86/20, 88/20 and may be used to access memory-based numeric operands, enabling convenient processing of numeric arrays, structures, based variables, etc.

NDP routines may also be written in PL/M-86, Fortran-86, and Pascal-86, Intel's high-level languages for the 86/20 and 88/20. These languages provide the programmer with access to many numeric facilities while reducing the programmer's need to understand the architecture of the system.

I/O PROCESSOR

- Complete I/O capability for iAPX product lines
- Memory-based communication with CPU
- Dual channels with full independent register files
- Allows mixed interface of 8/16 bit peripherals to 8/16 bit processor buses
- Supports local or remote I/O processing with I/O oriented instructions set
- Flexible, intelligent channel functions including: translation, search, word assembly/disassembly and DMA from port to memory, port to port or memory to memory

The IOP performs the function of an intelligent I/O channel for the iAPX product line and with its processing power can remove I/O overhead from the iAPX 86, 88, 188, 186, 286, 432. It may operate completely in parallel with a host processor giving dramatically improved performance in I/O intensive applications. The IOP provides two I/O channels, each supporting a transfer rate up to 1.25 megabyte/sec at the standard clock frequency of 5 MHz. Memory based communication between the IOP and CPU enhances system flexibility and encourages software modularity, yielding more reliable, easier to develop systems. The iAPX architecture provides for multiple IOP's in a system to incrementally increase performance.

The IOP continues the trend of simplifying the CPU's "view" of I/O devices by off loading another level of control from the CPU (Figure 10). The CPU performs an I/O operation by depositing a message in memory that describes the function to be performed; the IOP reads the message, carries out the operation and notifies the CPU when it has finished. All I/O devices appear to the CPU as transmitting and receiving complete blocks of data; the IOP can make both byte- and word-level transfers invisible to the CPU. The IOP assumes all device controller overhead, performs both programmed and DMA transfers, and allows recovery from "soft" I/O errors without CPU intervention; all of these activities may be performed while the CPU is executing other tasks.

In the remote configuration, one or more IOPs share a common system bus with the CPU (Figure 11). Access to this bus is controlled by 8289 Bus Arbiters. The IOP's I/O bus, however, is physically separated from the CPU in the remote configuration. Two IOPs can share the local I/O bus. Any number of remote IOPs may be contained in a system, configured in remote clusters of one or two. The local I/O bus need not be the same physical width as the shared system bus, allowing an IOP, for example, to interface 8-bit peripherals to an 86/10, 88/10. In the remote configuration, the IOP can access local I/O devices and memory without using the shared system bus, thereby reducing system bus contention with the CPU.

Contention can further be reduced by locating the IOP's channel programs in the local I/O space. The IOP can then also fetch instructions without accessing the system bus. CPU/IOP communication blocks must be located in shared (global) system memory, however, so that both processors can access them. The remote configuration thus increases the degree to which an IOP and a CPU can operate in parallel and thereby increases a system's throughput potential.

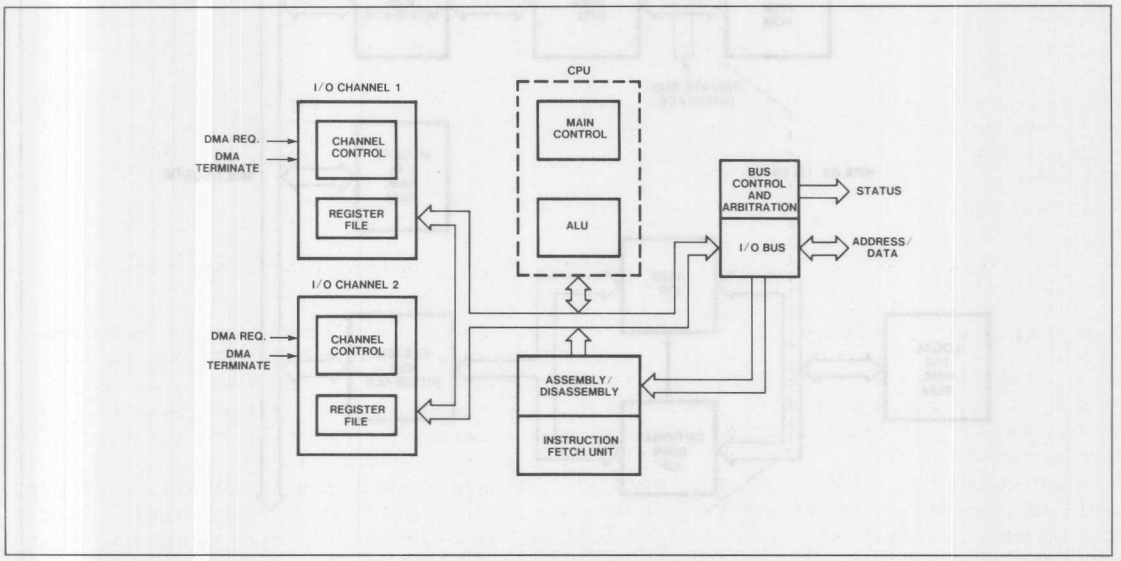


Figure 9. I/O Processor Block Diagram

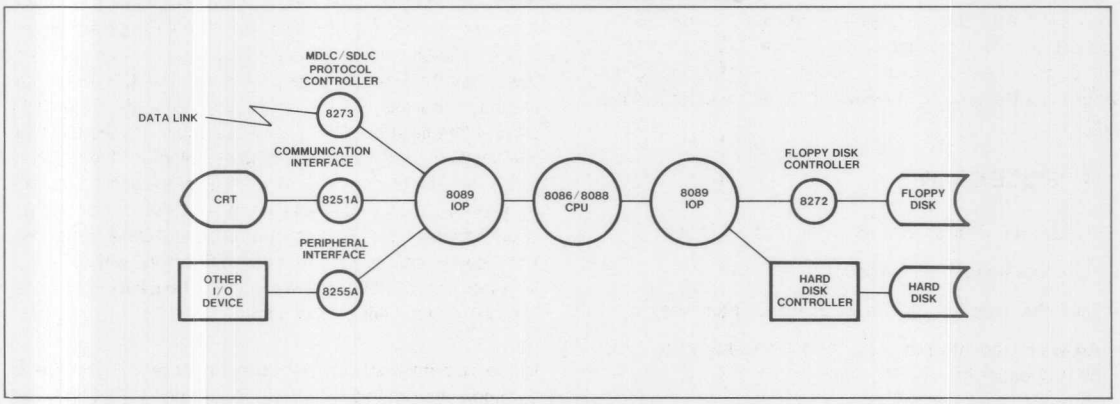


Figure 10. Typical I/O Control

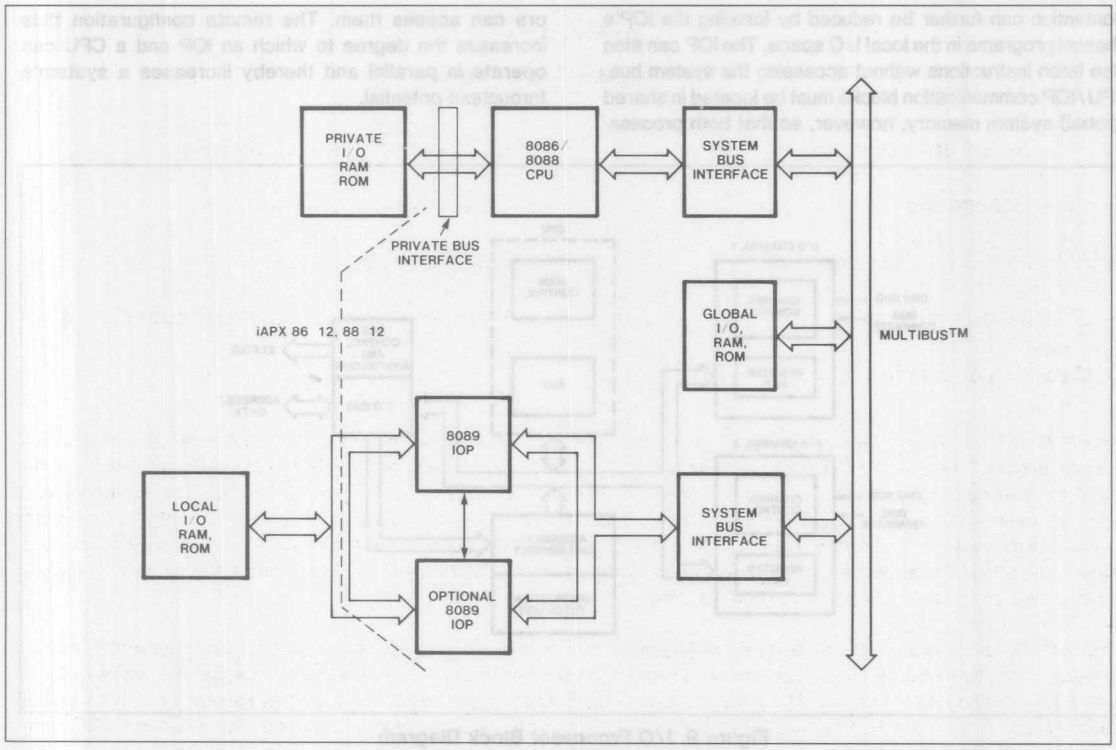


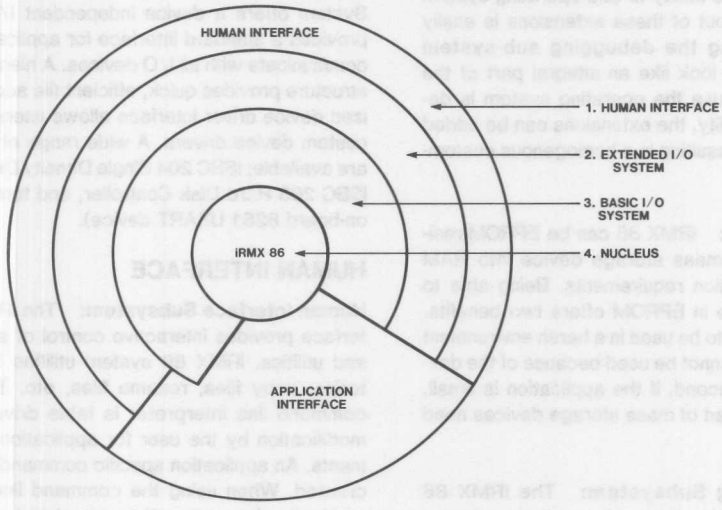
Figure 11. iAPX 86/12, 88/12 System

iRMX 86™ Operating System

- **Structured environment**
- **User extensible**
- **Powerful error handlers**
- **EPROM or RAM based**
- **User configurable**
- **Comprehensive I/O system**
- **Extensive debugging aids**
- **Full human interface**
- **Real-time priority oriented scheduling**

The iRMX 86 Real-Time Operating System is an easy-to-use, sophisticated, software system which operates on Intel iAPX 86 processors. The iRMX 86 Operating System extends the iAPX 86 architecture providing a structured, efficient environment for application programs. Services provided by the iRMX 86 Operating System include facilities for concurrent program execution, resource and information sharing, servicing asynchronous events, and interactive control over system resources and utilities. In addition, the iRMX 86 Operating System provides all major real-time facilities including priority-based system resource allocation; the means to concurrently monitor and control multiple external events; real-time clock control; and interrupt handling and task dispatching.

Because the services provided by the operating system are user selectable, application specific operating systems can be created. The iRMX 86 operating system thus eliminates the need for custom operating system design and hence reduces development time costs and risk. The iRMX 86 Operating System is a full featured operating system including a device independent input/output sub-system, a human interface sub-system with command language interpreter and ASCII console interface, a terminal handler, and an interactive debugger.



iRMX 86™

The iRMX 86 Operating System provides users of the Intel iAPX 86 simple, easy-to-use tools for creating a wide range of application systems. The most important features of the iRMX 86 Operating System are:

Structured Environment: The iRMX 86 Operating System provides a consistent structure from application to application thus allowing experience gained on one system to be easily transferred to others. Often entire programs may be used in multiple applications.

User Configurable: iRMX 86 Operating System based applications may be configured from a wide range of facilities, selecting only those which meet the specific requirements of the application system. The resultant system contains only the modules necessary for its use, allowing the iRMX 86 Operating System to be cost effectively applied in a wide range of application environments from performance oriented industrial applications, to security conscious data processing systems. The iRMX 86 Operating System is constructed in a thoroughly modular manner with the full range of facilities being offered in library modules, allowing easy selection of the exact features required. This eliminates overhead that might have been incurred for facilities not required.

User Extensible: The iRMX 86 Operating System provides a framework in which to extend the system and have these extensions look like facilities provided by Intel. These extensions include not only the ability to add custom system calls, but also the ability to add operating system data structures. Check-out of these extensions is easily accomplished by using the debugging sub-system because the extensions look like an integral part of the operating system. Because the operating system is designed for user extensibility, the extensions can be added in a symmetric manner resulting in a homogenous customized operating system.

EPROM or RAM Based: iRMX 86 can be EPROM-resident or loaded from a mass storage device into RAM depending upon application requirements. Being able to place all of the software in EPROM offers two benefits. First, if the application is to be used in a harsh environment mass storage devices cannot be used because of the danger of contamination. Second, if the application is small, the expense and overhead of mass storage devices need not be incurred.

Extensive Debugging Subsystem: The iRMX 86 Operating System provides interactive software debugging. The debugger permits both register and memory examination and modification. Task-oriented debugging is accomplished by symbolically setting breakpoints according to tasks rather than absolute memory locations. When a breakpoint has occurred, all operating system lists can be viewed so the system state can be determined.

NUCLEUS

The iRMX 86 nucleus provides a foundation upon which a variety of application systems can be built. A wide range of multi-programming, multi-tasking, and real-time oriented facilities are included. Extensive task to task communication and control permits easy task synchronization and data transfer thus allowing concurrent applications greater control over their execution environment.

Real-Time Priority Oriented Scheduler: The iRMX 86 Scheduler insures that the highest priority task ready to execute is given system control because the scheduler recognizes 255 software priority levels. Also, the system supports 64 hardware priority levels allowing the application system to be responsive to its external environment.

Error Handling Sub-system: The iRMX 86 Operating System provides extensive error handling and reporting mechanisms. Both excessive system loading and user operator errors can be reported causing system debug time to be shortened. The flexibility of the error handling subsystem allows errors to be serviced directly by the user task or sent to a specific error handler.

I/O SYSTEM

Comprehensive I/O System: The iRMX 86 Operating System offers a device independent I/O system which provides a standard interface for application programs to communicate with all I/O devices. A hierarchical directory structure provides quick, efficient file access. A standardized device driver interface allows users to easily create custom device drivers. A wide range of standard drivers are available; iSBC 204 Single Density Diskette Controller, iSBC 206 Hard Disk Controller, and terminal handler (via on-board 8251 USART device).

HUMAN INTERFACE

Human Interface Subsystem: The iRMX 86 Human Interface provides interactive control of system resources and utilities. iRMX 86 system utilities include file directories, copy files, rename files, etc. The Intel-supplied command line interpreter is table driven allowing easy modification by the user for application specific requirements. An application specific command language can be created. When using the command line interpreter this application language will be translated so that the appropriate user program will be invoked.

8282/8283
OCTAL LATCH

- Address Latch for iAPX 86,88, MCS-80™, MCS-85™, MCS-48™ Families
- High Output Drive Capability for Driving System Data Bus
- Fully Parallel 8-Bit Data Register and Buffer
- Transparent during Active Strobe
- 3-State Outputs
- 20-Pin Package with 0.3" Center
- No Output Low Noise when Entering or Leaving High Impedance State

The 8282 and 8283 are 8-bit bipolar latches with 3-state output buffers. They can be used to implement latches, buffers, or multiplexers. The 8283 inverts the input data at its outputs while the 8282 does not. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with these devices.

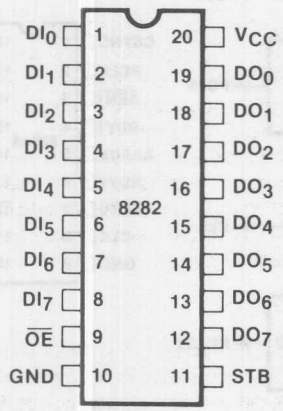


Figure 1. 8282 Pin Configuration

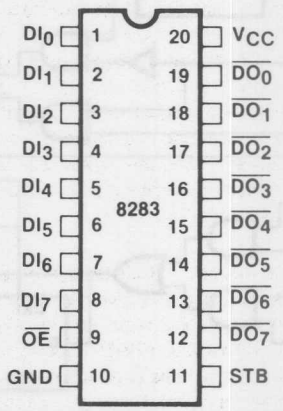


Figure 2. 8283 Pin Configuration

8284A
CLOCK GENERATOR AND DRIVER FOR iAPX 86, 88 PROCESSORS

- Generates the System clock for the iAPX 86, 88 Processors
- Uses a Crystal or a TTL Signal for Frequency Source
- Provides Local READY and Multibus™ READY Synchronization
- 18-Pin Package
- Single +5V Power Supply
- Generates System Reset Output from Schmitt Trigger Input
- Capable of Clock Synchronization with Other 8284As

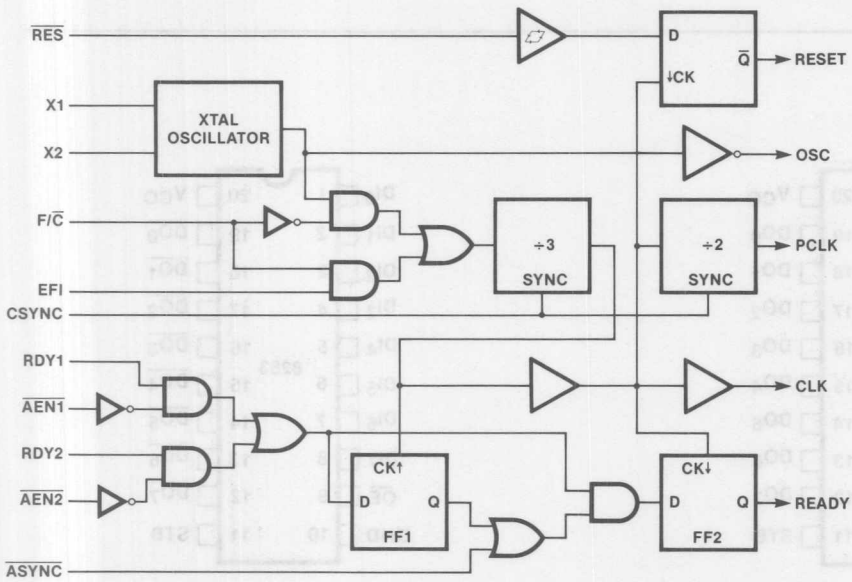


Figure 1. 8284A Block Diagram

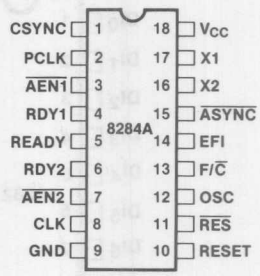


Figure 2. 8284A Pin Configuration

8286/8287
OCTAL BUS TRANSCEIVER

- **Data Bus Buffer Driver for iAPX 86,88, MCS-80™, MCS-85™, and MCS-48™ Families**
- **High Output Drive Capability for Driving System Data Bus**
- **Fully Parallel 8-Bit Transceivers**
- **3-State Outputs**
- **20-Pin Package with 0.3" Center**
- **No Output Low Noise when Entering or Leaving High Impedance State**

The 8286 and 8287 are 8-bit bipolar transceivers with 3-state outputs. The 8287 inverts the input data at its outputs while the 8286 does not. Thus, a wide variety of applications for buffering in microcomputer systems can be met.

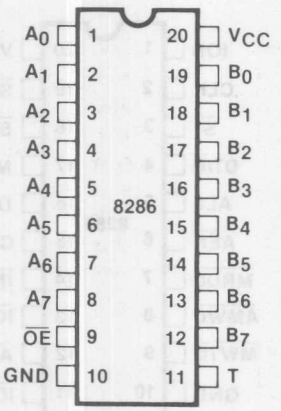


Figure 1. 8286 Pin Configuration

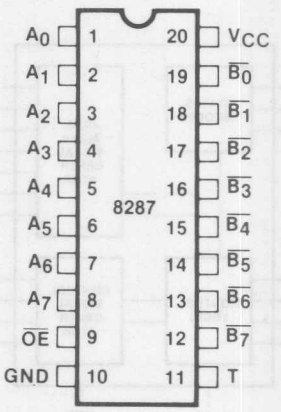


Figure 2. 8287 Pin Configuration

8288
BUS CONTROLLER
FOR iAPX 86,88 PROCESSORS

- **Bipolar Drive Capability**
- **Provides Advanced Commands**
- **Provides Wide Flexibility in System Configurations**
- **3-State Command Output Drivers**
- **Configurable for Use with an I/O Bus**
- **Facilitates Interface to One or Two Multi-Master Busses**

The Intel® 8288 Bus Controller is a 20-pin bipolar component for use with medium-to-large iAPX 86,88 processing systems. The bus controller provides command and control timing generation as well as bipolar bus drive capability while optimizing system performance.

A strapping option on the bus controller configures it for use with a multi-master system bus and separate I/O bus.

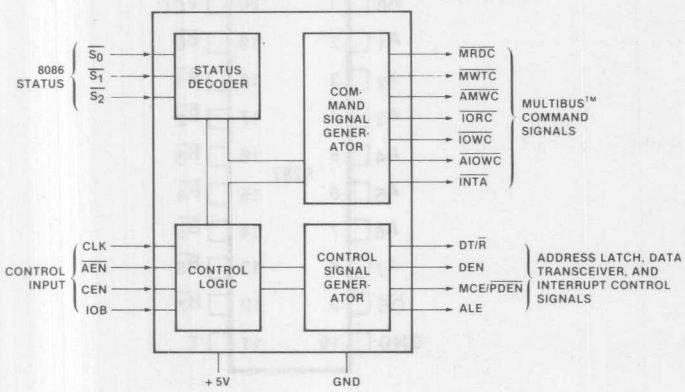


Figure 1. Block Diagram

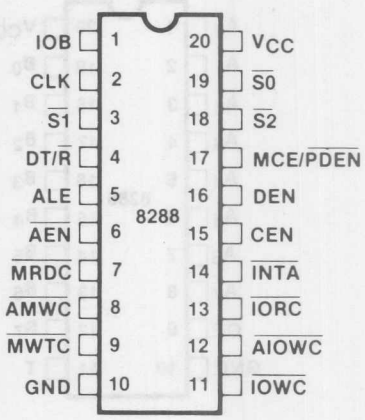


Figure 2. Pin Configuration

8289
BUS ARBITER

- Provides Multi-Master System Bus Protocol
- Synchronizes 8086/8088 Processors With Multi-Master Bus
- Provides Simple Interface With 8288 Bus Controller
- Four Operating Modes For Flexible System Configuration
- Compatible with Intel Bus Standard MULTIBUS™
- Provides System Bus Arbitration For 8089 IOP In Remote Mode

The Intel 8289 Bus Arbiter is a 20-pin, 5-volt-only bipolar component for use with medium to large iAPX 86,88 multimaster/multiprocessing systems. The 8289 provides system bus arbitration for systems with multiple bus masters, such as an 8086 CPU with 8089 IOP in its REMOTE mode, while providing bipolar buffering and drive capability.

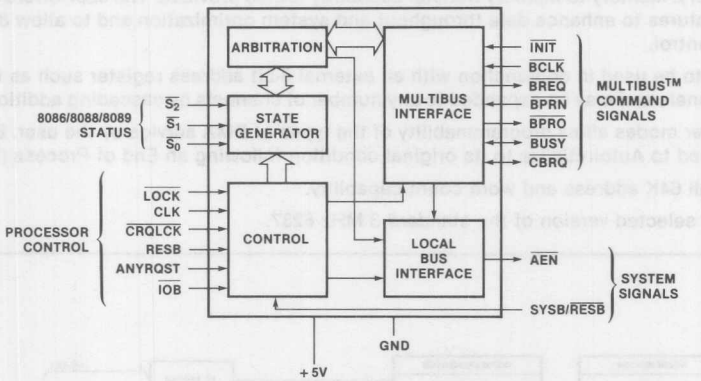


Figure 1. Block Diagram

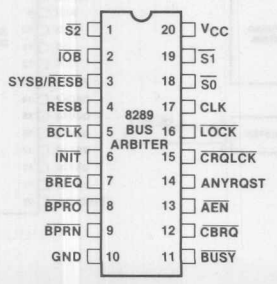


Figure 2. Pin Configuration

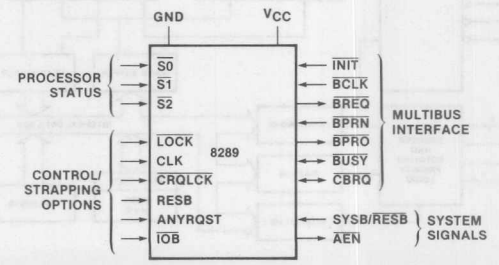


Figure 3. Functional Pinout

8237/8237-2
HIGH PERFORMANCE PROGRAMMABLE DMA CONTROLLER

- **Enable/Disable Control of Individual DMA Requests**
- **Four Independent DMA Channels**
- **Independent Autoinitialization of all Channels**
- **Memory-to-Memory Transfers**
- **Memory Block Initialization**
- **Address Increment or Decrement**
- **High Performance: Transfers up to 1.6M Bytes/Second with 5 MHz 8237-2**
- **Directly Expandable to any Number of Channels**
- **End of Process Input for Terminating Transfers**
- **Software DMA Requests**
- **Independent Polarity Control for DREQ and DACK Signals**

The 8237 Multimode Direct Memory Access (DMA) Controller is a peripheral interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information to or from the system memory. Memory-to-memory transfer capability is also provided. The 8237 offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.

The 8237 is designed to be used in conjunction with an external 8-bit address register such as the 8282. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips.

The three basic transfer modes allow programmability of the types of DMA service by the user. Each channel can be individually programmed to Autoinitialize to its original condition following an End of Process (EOP).

Each channel has a full 64K address and word count capability.

The 8237-2 is a 5 MHz selected version of the standard 3 MHz 8237.

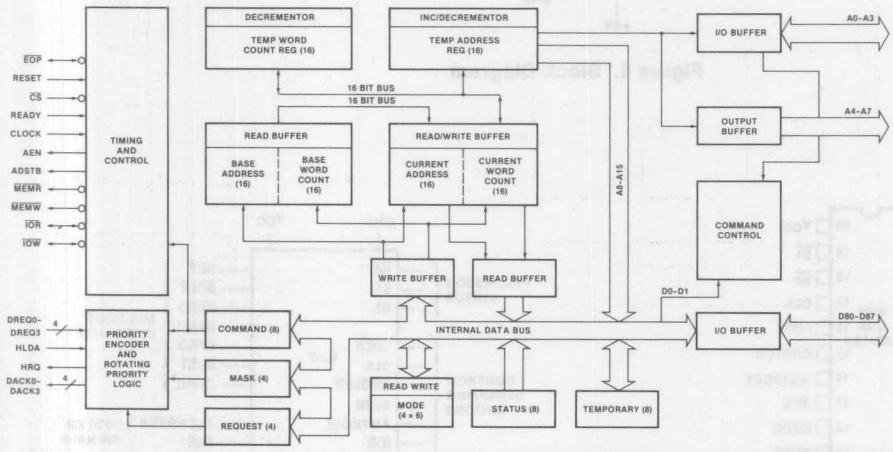


Figure 1. Block Diagram

Figure 2. Pin Configuration

8259A/8259A-2/8259A-8
PROGRAMMABLE INTERRUPT CONTROLLER

- iAPX 86, iAPX 88 Compatible
- MCS-80/85™ Compatible
- Eight-Level Priority Controller
- Expandable to 64 Levels
- Programmable Interrupt Modes
- Individual Request Mask Capability
- Single +5V Supply (No Clocks)
- 28-Pin Dual-In-Line Package

The Intel® 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single +5V supply. Circuitry is static, requiring no clock input.

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements.

The 8259A is fully upward compatible with the Intel® 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered).

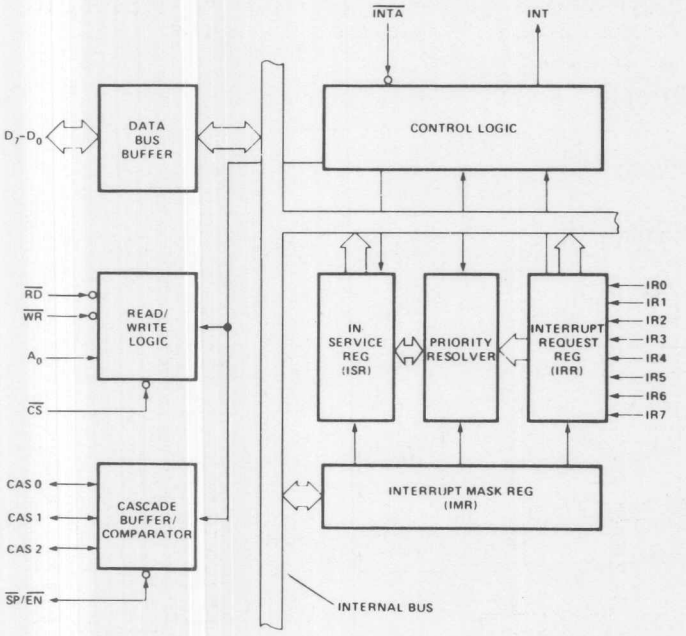


Figure 1. Block Diagram

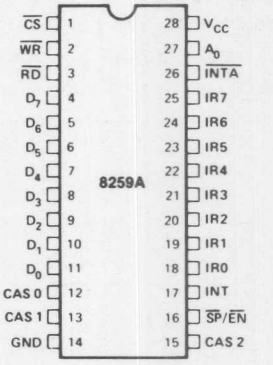


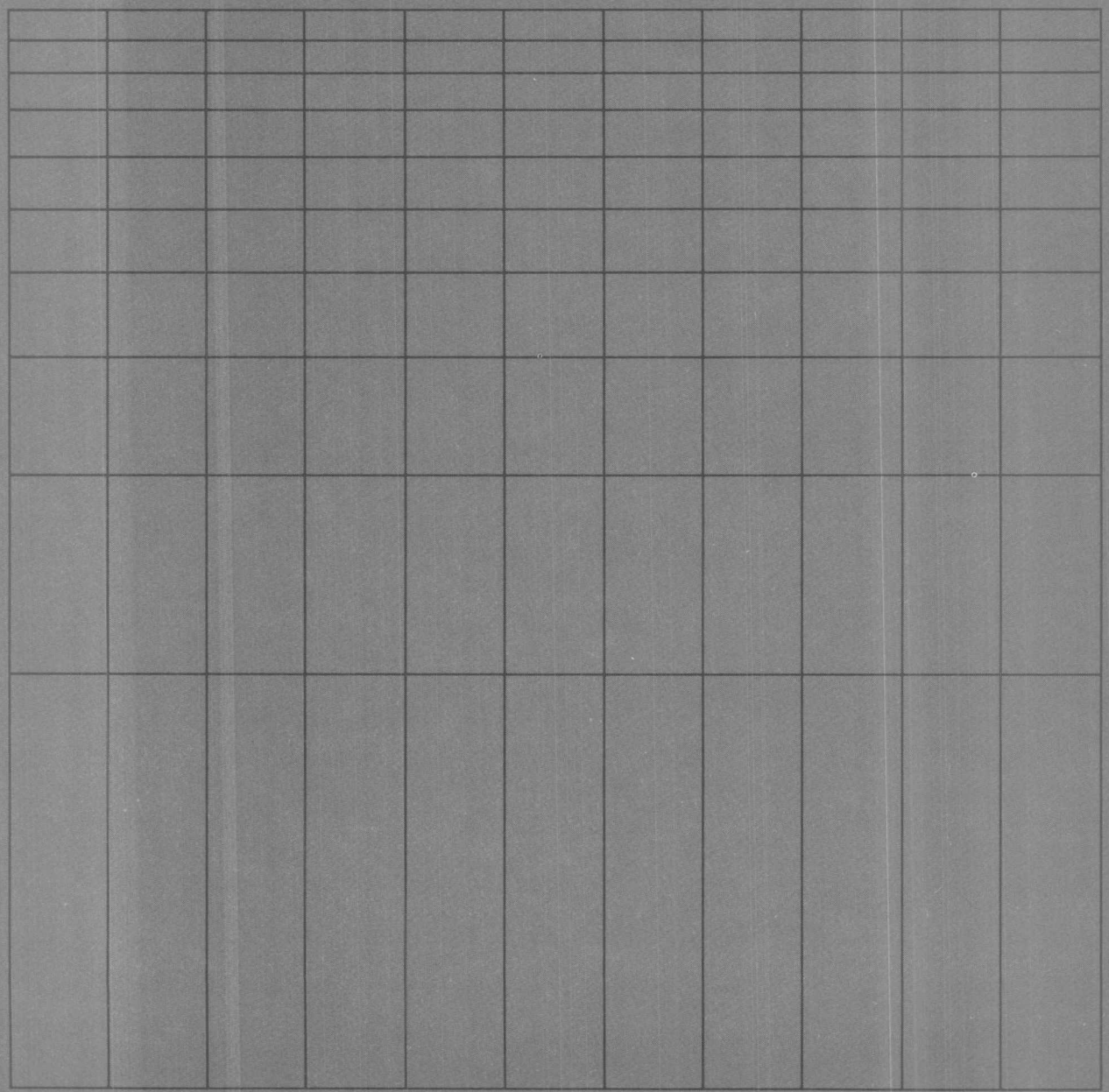
Figure 2. Pin Configuration

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A_0	COMMAND SELECT ADDRESS
$\overline{CS}$	CHIP SELECT
CAS2-CAS0	CASCADE LINES
$\overline{SP}/\overline{EN}$	SLAVE PROGRAM/ENABLE BUFFER
INT	INTERRUPT OUTPUT
$\overline{INTA}$	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

Table 1. Pin Names

Microsystem 80 Development Tools

3



MODEL 225
INTELLEC® SERIES II/85
MICROCOMPUTER DEVELOPMENT SYSTEM

- **Upgradable from previous Intellec® Series II systems**
- **Complete microcomputer development system for iAPX 86, iAPX 88, MCS-85™, MCS-80™, and MCS-48™**
- **High Performance 8085A-2 CPU, 64K bytes RAM memory, and 4K bytes ROM memory**
- **Self-test diagnostic capability**
- **Built-in interfaces for high speed paper tape reader/punch, printer, and universal PROM programmer**
- **Integral 250K byte floppy disk with total storage capacity expandable to over 2M bytes of floppy disk storage and 7.3M bytes of hard disk storage**
- **Powerful ISIS-II Disk Operating System with relocating macro assembler, linker, locater, and CRT-based editor CREDIT™**
- **Supports PL/M-80, PL/M-86, FORTRAN-80, BASIC-80, PASCAL-80 and COBOL-80 high-level languages**
- **Software compatible with previous Intellec® systems**
- **Compatible in distributed development system environments**

The Intellec Series II/85 Model 225 Microcomputer Development System is a performance enhanced complete microcomputer development system integrated into one compact package. The Model 225 includes a CPU with 64K bytes of RAM, 4K bytes of ROM, a 2000-character CRT, detachable full ASCII keyboard with cursor controls and upper/lower case capability, and a 250K byte floppy disk drive. Powerful ISIS-II Disk Operating System software allows the Model 225 to be used quickly and efficiently for assembling and debugging programs for Intel's iAPX 86, iAPX 88, MCS-85, or MCS-48 microprocessor families. ISIS-II performs all file handling operations for the user, leaving him free to concentrate on the details of his own application. When used with an optional in-circuit emulator (ICE) module, the Model 225 provides all the hardware and software development tools necessary for the rapid development of a microcomputer-based product. Optional storage peripherals provide over 2 million bytes of floppy disk, and 7.3 million of hard disk storage capacity.

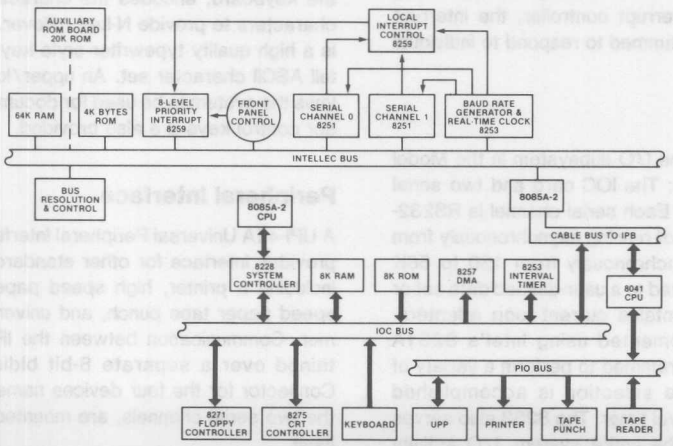


Figure 1. Intellec® Series II Model 225 Microcomputer Development System Block Diagram

FUNCTIONAL DESCRIPTION

Hardware Components

The Intellec Series II/85 Model 225 is a highly-integrated microcomputer development system consisting of a CRT chassis with a 6-slot cardcage, power supply, fans, cables, single floppy disk drive, and two printed circuit cards. A separate, full ASCII keyboard is connected with a cable. A block diagram of the Model 225 is shown in Figure 1.

CPU Cards—The master CPU card contains its own microprocessor, memory, I/O, interrupt and bus interface circuitry implemented with Intel's high technology LSI components. Known as the integrated processor card (IPC), it occupies the first slot in the cardcage. A second slave CPU card is responsible for all remaining I/O control including the CRT and keyboard interface. This card, mounted on the rear panel, also contains its own microprocessor, RAM and ROM memory, and I/O interface logic, thus, in effect, creating a dual processor environment. Known as the I/O controller (IOC), the slave CPU card communicates with the IPC over an 8-bit bidirectional data bus.

Expansion—Five remaining slots in the cardcage are available for system expansion. Additional expansion of 4 slots can be achieved through the addition of an Intellec Series II expansion chassis.

System Components

The heart of the IPB is an Intel NMOS 8-bit microprocessor, the 8085A-2 running at 4.0 MHz, 64K bytes of RAM memory are provided on the board using 16K RAMs. 4K of ROM is provided, preprogrammed with system bootstrap "self-test" diagnostics and the Intellec Series II/85 System Monitor. The eight-level vectored priority interrupt system allows interrupts to be individually masked. Using Intel's versatile 8259A interrupt controller, the interrupt system may be user programmed to respond to individual needs.

Input/Output

IPC Serial Channels—The I/O subsystem in the Model 225 consists of two parts: The IOC card and two serial channels on the IPC itself. Each serial channel is RS232-compatible and is capable of running asynchronously from 110 to 9600 baud or synchronously from 150 to 56K baud. Both may be connected to a user-defined data set or terminal. One channel contains current loop adapters. Both channels are implemented using Intel's 8251A USART. They can be programmed to perform a variety of I/O functions. Baud rate selection is accomplished through an Intel 8253 interval timer. The 8253 also serves as a real-time clock for the entire system. I/O activity through both serial channels is signaled to the system through a second 8259A interrupt controller, operating in a polled mode nested to the primary 8259A.

IOC Interface—The remainder of system I/O activity takes place in the IOC. The IOC provides interface for the CRT, keyboard, and standard Intellec peripherals including printer, high speed paper tape reader/punch, and universal PROM programmer. The IOC contains its own independent microprocessor, an 8080A-2. The CPU controls all I/O operations as well as supervising communications with the IPC. 8K bytes of ROM contain all I/O control firmware. 8K bytes of RAM are used for CRT screen refresh storage. These do not occupy space in Intellec Series II main memory since the IOC is a totally independent microcomputer sybsystem.

Integral CRT

Display—The CRT is a 12-inch raster scan type monitor with a 50/60 Hz vertical scan rate and 15.5 kHz horizontal scan rate. Controls are provided for brightness and contrast adjustments. The interface to the CRT is provided through an Intel 8275 single-chip programmable CRT controller. The master processor on the IPC transfers a character for display to the IOC, where it is stored in RAM. The CRT controller reads a line at a time into its line buffer through an Intel 8257 DMA controller and then feeds one character at a time to the character generator to produce the video signal. Timing for the CRT control is provided by an Intel 8253 interval timer. The screen display is formatted as 25 rows of 80 characters. The full set of ASCII character is displayed, including lower case alphas.

Keyboard—The keyboard interfaces directly to the IOC processor via an 8-bit data bus. The keyboard contains an Intel UPI-41A™ Universal Peripheral Interface, which scans the keyboard, encodes the characters, and buffers the characters to provide N-key rollover. The keyboard itself is a high quality typewriter style keyboard containing the full ASCII character set. An upper/lower case switch allows the system to be used for document preparation. Cursor control keys are also provided.

Peripheral Interface

A UPI-41A Universal Peripheral Interface on the IOC board provides interface for other standard Intellec peripherals including a printer, high speed paper tape reader, high speed paper tape punch, and universal PROM programmer. Communication between the IPB and IOC is maintained over a separate 8-bit bidirectional data bus. Connector for the four devices named above, as well as the two serial channels, are mounted directly on the IOC itself.

Control

User control is maintained through a front panel, consisting of a power switch and indicator, reset/boot switch, run-/halt light, and eight interrupt switches and indicators. The front panel circuit board is attached directly to the IPC, allowing the eight interrupt switches to connect to the primary 8259A, as well as to the Intellec Series II bus.

Integral Floppy Disk Drive

The integral floppy disk is controlled by an Intel 8271 single chip, programmable floppy disk controller. It transfers data via an Intel 8257 DMA controller between an IOC RAM buffer and the diskette. The 8271 handles reading and writing of data, formatting diskettes, and reading status, all upon appropriate commands from the IOC microprocessor.

MULTIBUS™ Interface Capability

All Intellec Series II/85 models implement the industry standard MULTIBUS protocol. The MULTIBUS enables several bus masters, such as CPU and DMA devices, to share the bus and memory by operating at different priority levels. Resolution of bus exchanges is synchronized by a bus clock signal derived independently from processor clocks. Read/write transfers may take place at rates up to 5 MHz. The bus structure is suitable for use with any Intel microcomputer family.

ICE-86™
8086 IN-CIRCUIT EMULATOR

Hardware in-circuit emulation

Full symbolic debugging

Breakpoints to halt emulation on a wide variety of conditions

Comprehensive trace of program execution, both conditional and unconditional

Disassembly of trace or memory from object code into assembler mnemonics

2K bytes of high speed ICE-86 mapped memory

Software debugging with or without user system

Handles full 1 megabyte addressability of 8086

Compound commands

Command macros

The ICE-86 module provides In-Circuit Emulation for the 8086 microprocessor and the iSBC 86/12 Single Board Computer. It includes three circuit boards which reside in Intellec® Microcomputer Development Systems. A cable and buffer box connect the Intellec system to the user system by replacing the user's 8086. Powerful Intellec debug functions are thus extended into the user system. Using the ICE-86 module, the designer can execute prototype software in continuous or single-step mode and can substitute blocks of Intellec system memory for user equivalents. Breakpoints allow the user to stop emulation on user-specified conditions, and the trace capability gives a detailed history of the program execution prior to the break. All user access to the prototype system software may be done symbolically by referring to the source program variables and labels.

ICE-88™
8088 IN-CIRCUIT EMULATOR

Hardware in-circuit emulation

Full symbolic debugging

Breakpoints to halt emulation on a wide variety of conditions

Comprehensive trace of program execution, both conditional and unconditional

Disassembly of trace or memory from object code into assembler mnemonics

2K bytes of high speed ICE-88 mapped memory

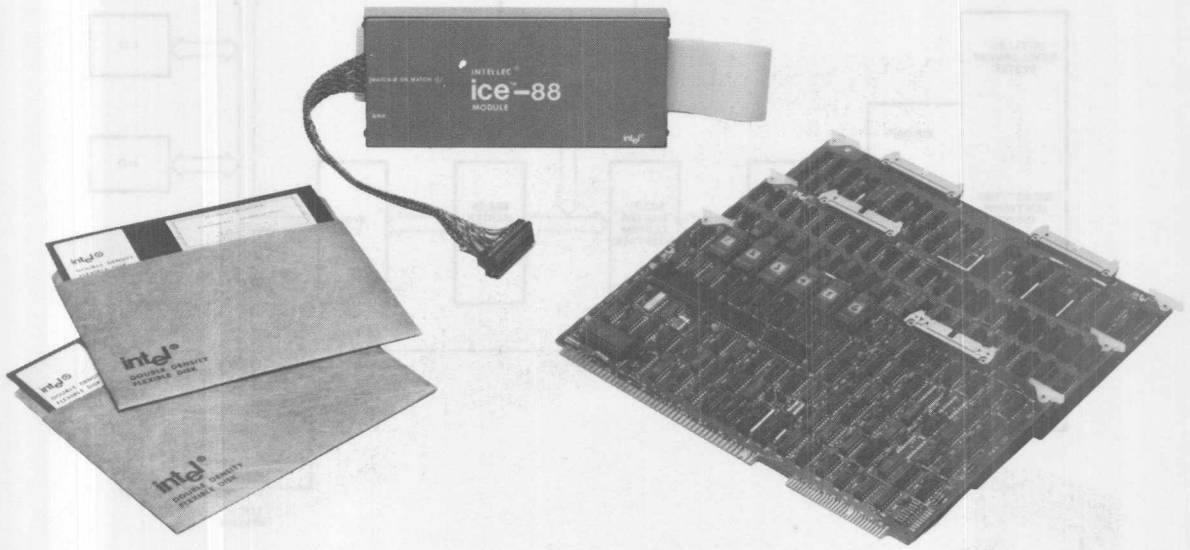
Software debugging with or without user system

Handles full 1 megabyte addressability of 8088

Compound commands

Command macros

The ICE-88 module provides In-Circuit Emulation for the 8088 microprocessor. It includes three circuit boards which reside in Intellec® Microcomputer Development Systems. A cable and buffer box connect the Intellec system to the user system by replacing the user's 8088. Powerful Intellec debug functions are thus extended into the user system. Using the ICE-88 module, the designer can execute prototype software in continuous or single-step mode and can substitute blocks of Intellec system memory for user equivalents. Breakpoints allow the user to stop emulation on user-specified conditions, and the trace capability gives a detailed history of the program execution prior to the break. All user access to the prototype system software may be done symbolically by referring to the source program variables and labels.



RBF-89
IOP REAL-TIME BREAKPOINT FACILITY

- **Debug IOP input/output processor programs in remote-mode iAPX86/11 microprocessor-based systems**
- **Load IOP programs into application system memory**
- **Execute programs in real time**
- **Halt execution at preset breakpoints**
- **Save debug environment on diskette**

The RBF-89 Real-Time Breakpoint Facility is a software package that aids in testing and troubleshooting systems designed around Intel iAPX 86/11 microprocessors. RBF-89 extends the debugging power of the ICE-86 In-Circuit Emulator into the IOP portion of your 86/11 system.

RBF-89 interrogates IOP registers, sets breakpoints in IOP programs, and performs its other functions by preparing special control blocks in application system memory and then issuing input/output channel-attention commands to the IOP in your system to perform these functions.

The RBF-89 package includes a host program that resides in Intellec development system RAM, where it serves as an extension of the ICE-86 emulator's software driver; a control program that resides in ICE-86 emulator memory and monitors IOP operations; and a utility program that resides in prototype memory, where it sets and removes the breakpoints.

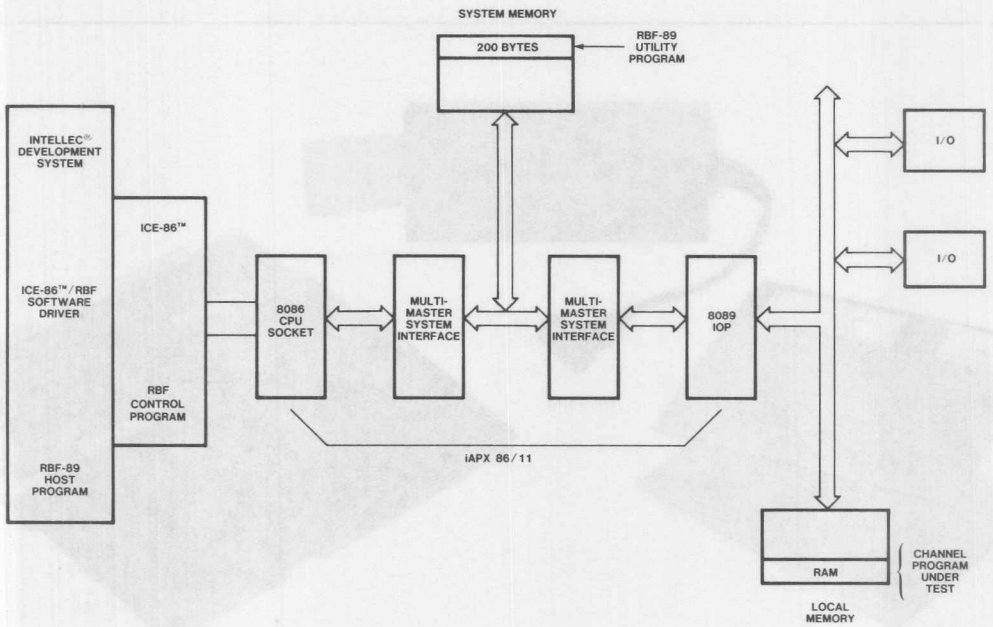


Figure 1. RBF 89 System Configuration in Remote Mode

iAPX 86, 88 SOFTWARE DEVELOPMENT PACKAGE (MDS-311)

- **Resident on Intellec® Microcomputer Development System**
- **Provides a complete set of iAPX 86 and iAPX 88 software development tools including:**
  - PL/M-86 high level programming language
  - ASM86 macro assembler for iAPX 86, 88 assembly language programming
  - LINK86 and LOC86 linkage and relocation utilities
  - CONV86 converter for conversion of 8080/8085 assembly language source code to iAPX 86, 88 assembly language source code
  - OH86 object-to-hexadecimal converter
  - LIB86 library manager
- **ICE™ Symbolic Debugging Fully Supported**

The iAPX 86, 88 software development package provides a set of software development tools for the iAPX 86 and the iAPX 88 microprocessors. The package is a complete set of iAPX 86 and iAPX 88 software development products running on the 8-bit based Intellec Model-800 and Series-II Microcomputer Development Systems. The software tools provided include the PL/M-86 high level programming language along with the powerful ASM-86 assembler. All of the necessary tools for linking and locating program modules are provided through the LINK86 and LOC86 utilities. The LIB86 Library Manager allows the user to maintain a library of iAPX 86 and iAPX 88 object modules to be included in various programs. The OH86 utility converts an iAPX 86, 88 absolute object module to a symbolic hexadecimal format. The CONV86 program allows users to convert 8080/8085 assembly language source programs into iAPX 86, 88 assembly language source programs.

The iAPX 86, 88 software development package executes on the 8-bit based Intellec Development Systems, either Model-800/Series-II or Series II/185. The package can also be executed on the 8-bit CPU in the Series-III Development System.

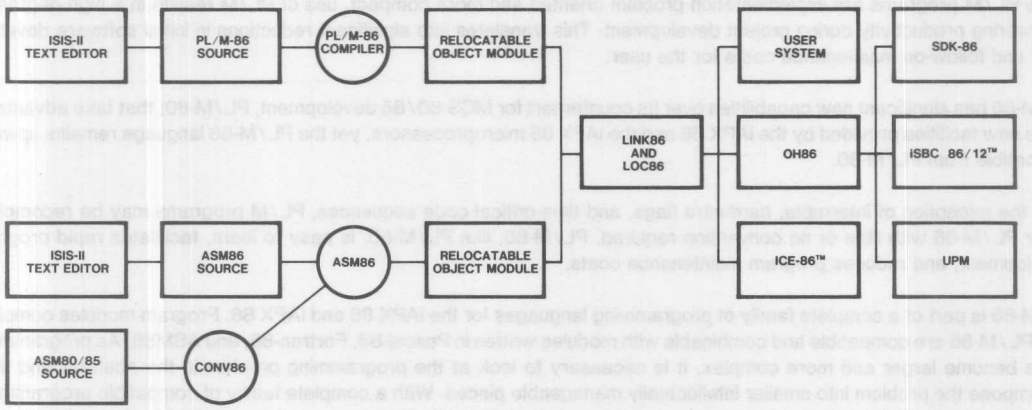


Figure 1. iAPX 86, 88 Development Using the iAPX 86, 88 Software Development Package

PL/M-86 SOFTWARE PACKAGE

- **iAPX 86 Resident PL/M High Level Programming Language**
- **Resident on iAPX 86 based INTELLEC Series III Microcomputer Development System for optimal compiler performance**
- **Allows user to achieve maximum benefits of iAPX 86, 88 capabilities**
- **Object compatible and linkable with PASCAL-86, FORTRAN-86, and ASM-86**
- **Upward compatible with PL/M-80 for software portability**
- **ICE™ symbolic debugging fully supported**
- **Implements Realmath™ standard for consistent and reliable results**
- **Supports iAPX 86/20, 88/20 numeric data processors**
- **Supports full extended addressing features of the iAPX 86 and the iAPX 88 microprocessors**
- **Code optimization assures efficient code generation and minimum application memory utilization**

PL/M-86 is a systems implementation language designed specifically for performing software development for the Intel iAPX 86 and iAPX 88 microprocessors. PL/M-86 includes features which allow the user high-level access to the microcomputer hardware, giving the designer close contact with and control over the microprocessor and its peripheral components. At the same time, however, it offers the ability to write code in English-like statements which can naturally express the program algorithm, resulting in efficient production of programs and programs that are easier to debug and maintain.

The PL/M-86 compiler efficiently converts free-form PL/M language statements into equivalent iAPX 86, 88 machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

Since PL/M programs are implementation problem oriented and more compact, use of PL/M results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-on maintenance costs for the user.

PL/M-86 has significant new capabilities over its counterpart for MCS-80/85 development, PL/M-80, that take advantage of the new facilities provided by the iAPX 86 and the iAPX 88 microprocessors, yet the PL/M-86 language remains upward compatible from PL/M-80.

With the exception of interrupts, hardware flags, and time-critical code sequences, PL/M programs may be recompiled under PL/M-86 with little or no conversion required. PL/M-86, like PL/M-80, is easy to learn, facilitates rapid program development, and reduces program maintenance costs.

PL/M-86 is part of a complete family of programming languages for the iAPX 86 and iAPX 88. Program modules compiled with PL/M-86 are compatible and combinable with modules written in Pascal-86, Fortran-86, and ASM86. As programming tasks become larger and more complex, it is necessary to look at the programming problem in the abstract and then decompose the problem into smaller intellectually manageable pieces. With a complete family of compatible programming languages for the iAPX 86, one can then implement each module in the language most appropriate to the task at hand.

FEATURES

Major features of the Intel PL/M-86 compiler and programming language include:

Supports Five Data Types

- Byte: 8-bit unsigned number
- Word: 16-bit unsigned number
- Integer: 16-bit signed number
- Real: 32-bit floating point number
- Pointer: 16-bit or 32-bit memory address indicator

Block Structured Language

- Permits use of structured programming techniques

Two Data Structuring Facilities

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of each: Arrays of structures or structures of arrays

Reentrant and Interrupt Procedures

- May be specified as user options

Relocatable and Linkable Object Code

- Permits PL/M-86 programs to be developed and debugged in small modules. These modules can be easily linked with other PL/M-86, Pascal-86, Fortran-86, or ASM-86 object modules and/or library routines to form a complete application system.

Built-In String Handling Facilities

- Operates on byte strings or word strings
- Six Functions: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, AND SET

Automatic Support for 8086 Extended Addressing

- Compiler options offer segmentation strategies for programs up to 1-Megabyte in size
- Language transparency for extended addressing

Support for ICE-86 Emulator and Symbolic Debugging

- Debug options allow inclusion of symbol table information in object modules for use with In-Circuit Emulation and symbolic debugging

Numerous Compiler Options

26 compiler options including:

- Conditional compilation
- Included file or copy facility
- Three levels of optimization
- Intra-module and inter-module cross reference
- Arbitrary placement of compiler and user files on any available combination of disk drives
- Quick Syntax checking compilation

BENEFITS

PL/M-86 is designed to be an efficient, cost-effective solution to the special requirements of iAPX 86, 88 Microcomputer Software Development, as illustrated by the following benefits of PL/M-86 use:

- Reduced Learning Effort—PL/M-86 is easy to learn and to use, even for the novice programmer
- Earlier Project Completion—Critical projects are completed much earlier than otherwise possible because PL/M-86, a structured high-level language, increases programmer productivity
- Lower Development Cost—Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.
- Increased Reliability—PL/M-86 is designed to aid in the development of reliable software (PL/M-86 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.
- Easier Enhancements and Maintenance—Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.
- Simpler Project Development—The PL/M-86 compiler runs on an Intellec Series III Microcomputer Development System and is part of a complete iAPX 86, 88 development solution. PL/M-86 is an integral part of a family of development tools which support your design project from source entry and language translation to software debug and hardware/software integration.

PASCAL-86 SOFTWARE PACKAGE

- **Resident on iAPX 86 based Intellec® Series III Microcomputer Development System for optimal performance**
- **Object compatible and linkable with PL/M-86, ASM-86 and FORTRAN-86**
- **ICE™ symbolic debugging fully supported**
- **Implements REALMATH standard for consistent & reliable results**
- **Supports iAPX 86/20, 88/20 Numeric Data Processors**
- **Strict implementation of ISO standard Pascal**
- **Useful extensions essential for microcomputer applications**
- **Separate compilation with type-checking enforced between Pascal modules**
- **MACRO language option**
- **Full run-time range-checking**
- **Removal of range-checking**

Pascal is a highly structured, block-oriented programming language which has become very popular as a microcomputer applications language. Its rigid structure encourages and enforces good programming techniques, which, combined with its high level of readability, helps produce more reliable and maintainable software.

Pascal-86 conforms to and implements the full ISO Draft Proposed Pascal standard. The standard language is enhanced with features needed to support microcomputer applications, such as separate compilation, interrupt handling and direct port I/O. To assist the development of portable software, a compiler option is available which flags all non-standard features.

The Pascal-86 compiler runs on the iAPX 86 Resident Intellec Series III Microcomputer Development System. The run-time system uses a well defined operating system interface to allow the Pascal 86 compiler and applications to run under Intel's development operating system under the iRMX 86 operating system, or, under a user-defined operating system. Program modules compiled under Pascal-86 are compatible and linkable with modules written in PL/M-86, ASM-86 or FORTRAN-86. With a complete family of compatible programming languages for the iAPX 86, 88 one can implement each module in the language appropriate for the task at hand.

Pascal-86 object modules contain the symbol record information for debugging programs using ICE-86.

The iAPX 86, 88 architecture was designed for high level languages, and Pascal-86 takes full advantage of it to generate efficient machine code. After debugging, the final production version can be further optimized using selected compiler options.

Pascal-86 uses the Intel REALMATH standard for arithmetic operations. (REALMATH is the de facto industry standard accepted by a majority of vendors and recommended to IEEE as the standard.) This not only provides support for Intel's numeric data processors such as the iAPX 86/20, it also provides compatibility with other languages in terms of a common interface and accuracy.

FEATURES:

—Includes all the language features defined in the ISO Draft Proposed Pascal standard.

—Supports required extensions for microcomputer applications.
- Interrupt handling
- Direct port I/O
- Structured constants

—Separate compilation extensions allow:
- Modular decomposition of large programs
- Linkage with other Pascal modules as well as PL/M-86, ASM-86 and FORTRAN-86.
- Enforcement of type-checking at LINK-time

—String handling procedures and functions are provided:
- Translate with a table
- Scan for characters or substrings
- Move, compare strings or substrings
- Defined as Pascal procedures for portability, implemented as built-in procedures for efficiency.

—Provides MACRO capability compatible with ASM-86.

—Supports numerous compiler options to control the compilation process, to include files, to flag non-standard Pascal statements and to control program listings and object modules.

Well defined and documented run-time operating system interfaces allow use with user-designed operating systems.

BENEFITS:

—Provides a standard Pascal for iAPX 86, 88 based applications.
- Pascal has gained wide acceptance as the portable application language for microcomputer applications
- It is being taught in many colleges and universities around the world
- It is easy to learn, originally intended as a vehicle for teaching computer programming
- Improves maintainability: Type mechanism is both strictly enforced and user extendable
- No machine specific language constructs

—Strict implementation of the proposed ISO standard for Pascal aids portability of application programs, e.g., a compile time option checks conformance to the standard making it easy to write portable programs.

—Pascal-86 extensions via predefined procedures for interrupt handling and direct port I/O make it possible to code on entire application in Pascal without compromising portability.

—Full Intel REALMATH standard is easy to use and provides consistent and reliable results.

—Provides run-time support for co-processors. All real-type arithmetic is performed on the 86/20 numeric data processor unit or software simulator. Run-time library routines, common between Pascal and other Intel languages (such as PL/M, FORTRAN and ASSEMBLY), permits consistently accurate results.

—Extended relocation and linkage support allows the user to link Pascal program modules with routines written in other languages for certain parts of his program. For example, real-time or hardware dependent routines written in ASM-86 or PL/M-86 can be automatically linked to Pascal routines, further extending the user's ability to write structured and modular programs.

—Pascal programs "talk" to the resident operating system using Intel's standard interface for all translators. This allows the user to replace the development operating system by his own interfaces in the final application.

—Compiler options can be used to control the program listings and object modules. While debugging, the user may generate additional information such as the symbol record information required and useful for debugging using ICE. After debugging, the production version may be optimized by removing this additional information.

—MACRO facility is used for conditional compilation and implementation of in-line sub-routines. Debugged procedures may be declared as MACRO'S and need not be expanded.

Microsystem 80
Single Board Computers

4

MULTIBUS™ SYSTEM BUS

- **Provides system bus interconnection for complete family of iSBC 80™ and iSBC 86™ products**
- **IEEE approval pending**
- **Built-in architecture for multiprocessing with 8-, 16-bit single board computers or both**
- **Support for high performance system configurations**
  - Multiple masters
  - Masters and intelligent slaves
  - Masters and slaves
- **Data rates up to 5 million 8- or 16-bit transfers per second**
- **One megabyte direct addressability; planned expansion to 16 Mbytes**
- **Full line of compatible products allows wide selection of performance options and easy system expandability**

One of the most important elements in a computer system is the bus structure that holds all the hardware components together. This bus structure contains the necessary signals to allow the various system components to interact with each other, i.e., it allows memory and I/O data transfers, direct memory accesses, generation of interrupts, etc.

The MULTIBUS system bus is the flexible bus structure used to interface the family of Intel's iSBC 80 and iSBC 86 products which include 8- and 16-bit single board computers, memory expansion boards, digital and analog I/O boards and peripheral controllers. It supports direct addressability up to one megabyte through 20-bit addresses and 8- and 16-bit data transfers. Plans are in place to increase the addressability to 24-bits to support 16 megabytes of directly addressable memory.

The bus structure is built upon the master-slave concept where the master device in the system takes control of the MULTIBUS interface and the slave device, upon decoding its address, acts upon the command provided by the master. This handshake between master and slave devices allows modules of different speeds to use the MULTIBUS interface and allows data rates up to five million transfers per second (bytes or words).

Another important MULTIBUS feature is the ability to connect multiple master modules for multiprocessing configurations. The MULTIBUS interface provides control signals for connecting multiple masters either in a daisy-chain priority fashion or in parallel. With this latter arrangement, up to sixteen masters may share MULTIBUS resources.

MULTIMODULE™ BOARDS AND THE NEW iSBX™ BUS

- **Offers incremental expansion on-board A new generation of MULTIBUS™ compatible SBC boards**
- **Three introductory MULTIMODULES™**
  - iSBX 350™ Parallel I/O
  - iSBX 351™ Serial I/O
  - iSBX 332™ Floating Point Math Processor
- **Multimodule connector available for custom MULTIMODULE™ board design**

THE iSBX™ BUS

The MULTIMODULE board concept offers a unique design approach to board level users. To date, the designer has had a broad range of single board computers and expansion boards joined together on the MULTIBUS interface. The iSBX MULTIMODULE boards bring a new concept to expansion, providing a product family of smaller modules that can be plugged directly onto the single board computer. In short, the user may now tailor his application directly on-board the single board computer at a minimal cost. In addition, the iSBX boards offer maximum performance because they are tightly coupled to the microprocessor through the iSBX bus.

The common interface between the new single board computer and iSBX MULTIMODULE boards is the new iSBX bus. The iSBX bus is derived directly from the on-board CPU bus and, as such, an iSBX board plugged into the iSBX bus becomes an integral element of the single board computer. The physical interface between the single board computer and the iSBX MULTIMODULE board is a unique connector designed specifically for the iSBX bus. The iSBX bus is brought to a female iSBX bus connector on the single board computer and mates with its male equivalent resident on the iSBX board (see Figure 1).

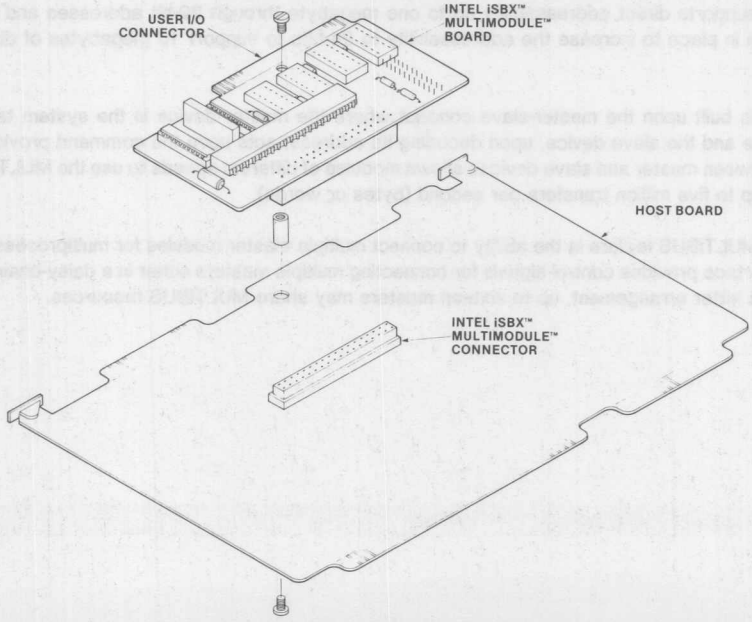


Figure 1. Connection of iSBX™ MULTIMODULE™ Board to Single Board Computer

iSBC 86/12A™
SINGLE BOARD COMPUTER

8086 16 bit HMOS microprocessor central processor unit

32K-bytes of dual-port read/write memory expandable on-board to 64K-bytes with on-board refresh

Sockets for up to 16K-bytes of read only memory expandable on-board to 32K-bytes

System memory expandable to 1 megabyte

24 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators

Programmable synchronous/ asynchronous RS232C compatible serial interface with software selectable baud rates

Two programmable 16-bit BCD or binary timers/event counters

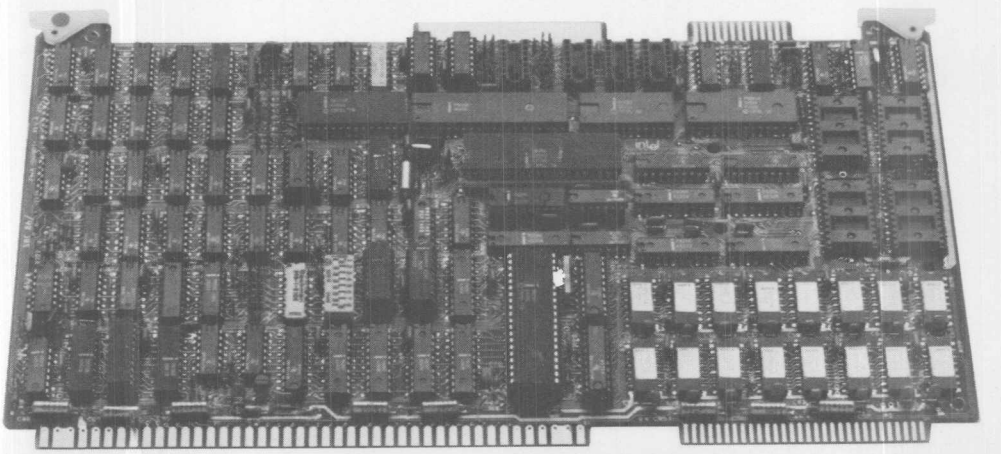
9 levels of vectored interrupt control, expandable to 65 levels

Auxiliary power bus and power fail interrupt control logic for read/write memory battery backup

MULTIBUS interface for multimaster configurations and system expansion

Compatible with iSBC 80 family single board computers, memory, digital and analog I/O, and peripheral controller boards

The iSBC 86/12A Single Board Computer is a member of Intel's complete line of OEM microcomputer systems which take full advantage of Intel's LSI technology to provide economical self-contained computer based solutions for OEM applications. The iSBC 86/12A board is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the board. Full MULTIBUS interface logic is included to offer compatibility with the Intel OEM Microcomputer Systems family of Single Board Computers, expansion memory options, digital and analog I/O expansion boards and peripheral controllers.



ADDITIONAL INTEL RELATED PRODUCTS

Additional information on these products can be obtained from Intel's 1980 Component Data Catalog and 1980 Systems Data Catalog.

RANDOM ACCESS MEMORY

2109 Family	8,192×1-Bit Dynamic RAM
2115A, 2125A Family	High Speed 1K×1-Bit Static RAM
2115H, 2125H Family	High Speed 1K×1-Bit Static RAM
2117 Family	16,384×1-Bit Dynamic RAM
2147	4096×1-Bit Static RAM
2147H	High Speed 4096×1-Bit Static RAM
2148	1024×4-Bit Static RAM
2148H	1024×4-Bit Static RAM
2149H	1024×4-Bit Static RAM

READ ONLY MEMORY AND MEMORY SUPPORT

2708	8K (1K×8) UV Erasable PROM
2716	16K (2K×8) UV Erasable PROM
2732	32K (4K×8) UV Erasable PROM
2758	8K (1K×8) UV Erasable Low Power PROM
3205	High Speed 1 Out of 8 Binary Decoder
3404	High Speed 6-Bit Latch
3242	Address Multiplexer and Refresh Counter for 16K Dynamic RAMs
3628	8K (1K×8) Bipolar PROM
3628A	8K (1K×8) Bipolar PROM
3636	16K (2K×8) Bipolar PROM

MAGNETICS

7230	Current Pulse Generator for Bubble Memories
7242	Dual Formatter/Sense Amplifier for Bubble Memories
7250	Coil Pre-Driver for Bubble Memories
7254	Quad VMOS Drive Transistors for Bubble Memories
BPK71	Bubble Memory Prototype Kit
IMB100	1 Megabit Bubble Memory Development Board
IMB101	1 Megabit Bubble Memory Board

TELEPHONY AND SIGNAL PROCESSING

2910A	PCM Codec—μLaw, 8-Bit Companded A/D and D/A Converter
2911A	PCM Codec—A Law, 8-Bit Companded A/D and D/A Converter
2912 Family	PCM Line Filters